COMPUTE!'s FIRST BOOK OF IBM

The best games, applications, tutorials, and utilities from COMPUTE! Publications. Play chess against your PC, create intricate graphics, examine a disk, escape from Martians, and learn some programming tricks. For users of the IBM PC, PC XT, and PCjr personal computers.

A **COMPUTE! Books** Publication

$14.95

COMPUTE!'s FIRST BOOK OF IBM

COMPUTE! Publications, Inc. abc
One of the ABC Publishing Companies
Greensboro, North Carolina

The following articles were originally published in *COMPUTE!* magazine, copyright 1984, COMPUTE! Publications, Inc.:
"PC Monochrome Graphics" (November); "IBM Personalized Form Letters" (December).

The following articles were originally published in *COMPUTE!'s PC & PCjr* magazine, copyright 1984, COMPUTE! Publications, Inc.:
"Martian Prisoner" (March); "Spelling Bee" (March); "The 8088 Microprocessor: Brains of PC & PCjr" (March); "Word Hunt" (March); "Choosing a Display for your PC/PCjr" (April); "Getting Started with a Disk Drive" (April); "The Screen Machine: Graphics Made Easy" (April); "Aardvark Attack" (June); "Gradebook" (June); "Munchmath" (June); "Notemaker" (June); "PC/PCjr PEEKs and POKEs" (June); "States and Capitals Tutor" (June); "Beginning BASIC: READ, DATA, and RESTORE" (July); "Calendar Maker" (July); "Custom Characters" (July); "Customizing the Function Keys" (July); "Laser Barrage" (July); "Bowling Champ" (August); "Quickversi" (August); "Simplified Date and Time" (August); "Super Directory" (August); "BASIC Hints" (September—originally titled "Beginning BASIC: Variables, Arrays, and PRINT Using"); "Experiments in Machine Language" (September); "TERMPlus" (September); "Creating Animation with PUT and GET" (October—originally titled "Animation Made Easy"); "Sculpt-a-Shape Graphics Editor" (October).

The following articles were originally published in *COMPUTE!* magazine, copyright 1985, COMPUTE! Publications, Inc.:
"IBM Rebound" (February); "IBM BASIC's Undocumented SHELL Command" (April); "Home Financial Calculator" (May); "IBM Disk Rx" (May); "Unlocking IBM BASIC Programs" (June); "Chess" (September).

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-87455-010-6

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. IBM PC, PC XT, and PCjr are trademarks of International Business Machines, Inc.

Contents

Foreword

Maybe you bought your IBM hoping you could use it at home for personal use as well as a business tool. Or maybe you wanted your kids to become computer literate, and since you had a PC at work, the PCjr seemed like a good choice. Whatever your reason, you chose a powerful machine. And COMPUTE! Publications has been supporting it continually in our magazines and books since 1983. This latest publication for IBM personal computers, *COMPUTE!'s First Book of IBM*, has something for every IBM user.

With thinking games (like "Chess" and "Quickversi"), educational games ("Spelling Bee," Word Hunt," and "States and Capitals Tutor"), and games that require quick reflexes ("Bowling Champ," "Aardvark Attack," and "Rebound"), *COMPUTE!'s First Book of IBM* is sure to give every game player the fun and challenge he or she wants in a good computer game.

Did you think the built-in terminal program *TERM* was the only communications software you needed for your PCjr, only to be surprised to discover that it didn't upload or download? Now just add the modifications found in Chapter 1, and transform *TERM* into "TERMPlus," a terminal program that will turn your PCjr into a complete communications terminal with uploading and downloading capabilities.

Do you find yourself having to enter and reenter the date and time each time you turn on your IBM because you can't get the format just right? "Simplified Date and Time" makes the task easier.

Perhaps you've always wanted to write your own programs with complex graphics? Learn how to create your own characters and make intricate patterns on a monochrone display. Use "The Screen Machine" and "Sculpt-a-Shape Graphics Editor" to design, create, and save vivid graphics to add to your own programs or simply to recall and admire.

As with all COMPUTE! publications, the writing is clear and concise. And all the programs have been debugged and are ready to type in. We've even included "The Automatic Proofreader," an aid that checks your typing as you enter each

program line, making program entry a snap. Or, if you'd prefer not to type in the programs, you can purchase a disk with all the programs from *COMPUTE!'s First Book of IBM* by using the coupon in the back of this book or by calling toll-free 1-800-334-0868.

CHAPTER 1

Getting More from Your IBM

Simplified Date and Time

Michael A. Covington

Make your DOS booting a little easier with this utility program. It works with standard Disk BASIC on the PC or PCjr.

Quick, now, how do you say 4:30 p.m. in 24-hour notation? Is it 15:30:00 or 16:30:00 (or 00:16:30)? And will the Fourth of July be 06-04-84 or 07-04-84?

As supplied by IBM, the Disk Operating System (1.0, 1.1, 2.0, or 2.1) requires you to answer questions like these every time you turn on your PC or PCjr. It's tempting to just hit Enter and let the computer think that a certain mystical Tuesday in 1981 (or whenever your version of DOS came off the assembly line) is going to last forever. But if you yield to this temptation, the date stamps that DOS records for you are meaningless, and you'll never be able to tell which version of your latest masterpiece is the current one.

A Simpler Way

Enter "DATETIME.BAS," a program that does what DOS should have done. With DATETIME, you can call the Fourth of July "July 4, 1984"—or JUL 4 84 or 7-4-84 or a host of other things. Similarly, 4:30 p.m. is 4:30 p.m.; if you just type 4:30, the computer asks whether you mean a.m. or p.m. (You're still free to type 16:30 or 16:30:00 if 24-hour time really is more familiar to you.) If you type something that the computer can't understand, it gives you a (partial) list of acceptable formats and invites you to try again.

To get all this convenience, type in the accompanying program listing and call it DATETIME.BAS. **(Warning:** Save it before you run it! It kicks you out of BASIC at the end of every run.) DATETIME.BAS should reside, along with DOS and the BASIC interpreter, on the disk from which you boot.

If your boot disk contains a file called AUTOEXEC.BAT, edit that file (using EDLIN or another editor) to take out the DATE and TIME commands, if any, and insert the command

BASIC DATETIME

If there is no AUTOEXEC.BAT file, create one containing BASIC DATETIME as its only line. This will boot up DATETIME and BASIC whenever you switch on the computer. If you purchased the *COMPUTE!'s First Book of IBM* disk, the AUTOEXEC.BAT is on the disk (just follow the directions in Appendix C to add the system files to the disk).

Notes on EDLIN and AUTOEXEC.BAT

If you've never used EDLIN to create an AUTOEXEC.BAT file, here's a crash course.

First of all, AUTOEXEC.BAT is a file which executes automatically whenever you boot up the computer. The file must be on the DOS disk you use to boot up. The idea of AUTOEXEC.BAT is to save you the trouble of typing in the same commands every time you start a session with the computer. For instance, if you always find yourself typing BASIC to load the BASIC interpreter each time you boot up, you can create an AUTOEXEC.BAT file to type BASIC for you. Likewise, you can AUTOEXEC.BAT to load and run DATETIME.BAS for you, too.

EDLIN is a utility program for editing source files such as AUTOEXEC.BAT. EDLIN comes on your DOS system disk. To load EDLIN, insert the DOS disk in drive A, boot up the computer, and enter EDLIN AUTOEXEC.BAT at the DOS prompt. If there is no AUTOEXEC.BAT file already on the disk, the next thing you'll see is the message "New File" and an asterisk. The asterisk is the standard prompt in EDLIN.

Now type an I. This puts EDLIN in Insert mode so you can enter new commands. You'll see a numeral 1 next to the asterisk; this is the line number of the command you're about to enter (just like a BASIC program). Next, type BASIC DATETIME (or BASICA DATETIME if you use BASICA on a PC), and press Enter. This command means that AUTOEXEC.BAT will first load BASIC, and then load and run the program DATETIME (so make sure it's on the same disk).

Next, press Ctrl-Break (the Control and Scroll Lock keys). This exits EDLIN's Insert mode. To store the new AUTOEXEC.BAT file on your DOS disk, just type an E at the asterisk prompt. The AUTOEXEC.BAT file is created on your disk, EDLIN ends, and you're back in DOS.

Now whenever you boot up this DOS disk, BASIC will load itself, DATETIME will load and run itself, and the com-

puter will return to DOS after you enter the date and time. If you'd rather end up in BASIC or (on the PC) in BASICA, use EDLIN to enter the appropriate command to the AUTO-EXEC.BAT file.

For more details on using EDLIN, see Chapter 7 in the IBM DOS manual.

Datetime

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
NL 10 REM DATETIME.BAS
CL 20 DEFINT A-Z
KG 30 REM --- Set numeric keypad to NUM LOCK mode
IE 40 DEF SEG=0 : POKE &H417,(PEEK(&H417) OR &H20
      )
JB 50 '
MG 60 REM --- Accept date and time
JD 70 '
PD 80 ON ERROR GOTO 1230 ' routine to handle bad
      input
AF 90 '                      (any error condition
      will also invoke it)
HK 100 BEEP : PRINT
LB 110 PRINT "Today's date?  ";
AG 120 LINE INPUT INPTLN$
KN 130 GOSUB 1030 ' character set translation
IC 140 GOSUB 1140 ' FIRSTWD$ is now month; INPTLN
       $ is rest of input line
GB 150 DT$=""
MM 160 RESTORE
DA 170 IF LEN(FIRSTWD$)>3 THEN FIRSTWD$=LEFT$(FIR
       STWD$,3)
LD 180 FOR J=1 TO 2         ' recognize the month
PH 190   FOR I=1 TO 12
CI 200     READ MONTH$
KC 210     IF FIRSTWD$=MONTH$ THEN DT$=MID$(STR$(
       I),2)
LE 220                        ' (STR$ numbers ha
       ve a leading blank)
AG 230    NEXT I
OK 240 NEXT J
HP 250 '
PM 260 DATA "JAN","FEB","MAR","APR","MAY","JUN"
DB 270 DATA "JUL","AUG","SEP","OCT","NOV","DEC"
OO 280 DATA "1","2","3","4","5","6","7","8","9","
       10","11","12"
HH 290 '
JE 300 IF DT$="" THEN 1230 ' month not successful
       ly found
HI 310 '
```

```
QM 320 GOSUB 1140  ' FIRSTWD$ is now day of month
HG 330 DAY = VAL(FIRSTWD$)
CH 340 IF DAY<1 OR DAY>31 THEN 1230
HA 350 DT$=DT$+"/"+MID$(STR$(DAY),2)
HC 360 '
LN 370 GOSUB 1140  ' FIRSTWD$ is now year
OB 380 YEAR = VAL(FIRSTWD$)
FP 390 IF YEAR<1900 THEN YEAR=YEAR+1900
AD 400 IF YEAR<1980 OR YEAR>2099 THEN 1230
KE 410 DT$=DT$+"/"+MID$(STR$(YEAR),2)
HL 420 '
HN 430 '
IB 440 REM -- Accept time
HB 450 '
MI 460 ON ERROR GOTO 1390
CL 470 PRINT : PRINT "Time of day?   ";
BF 480 LINE INPUT INPTLN$
LM 490 GOSUB 1030 ' character set translation
HI 500 '
NC 510 GOSUB 1140 ' FIRSTWD$ is now hours
GC 520 IF LEFT$(FIRSTWD$,1)<"0" OR LEFT$(FIRSTWD$
       ,1)>"9" THEN 1390
FF 530 HRS=VAL(FIRSTWD$)
IB 540 IF HRS=12 THEN HRS=0    ' idiosyncrasy of 1
       2 a.m. and p.m.
HP 550 IF HRS>23 THEN 1390
HE 560 '
LK 570 GOSUB 1140 ' FIRSTWD$ is now minutes
HO 580 IF LEFT$(FIRSTWD$,1)<"0" OR LEFT$(FIRSTWD$
       ,1)>"9" THEN 1390
HB 590 MINS=VAL(FIRSTWD$)
PG 600 IF MINS>59 THEN 1390
HL 610 '
FA 620 GOSUB 1140 '  FIRSTWD$ is now either secon
       ds or a.m./p.m.
GL 630 SECS=0
IN 640 IF FIRSTWD$="" THEN 760
IK 650 IF LEFT$(FIRSTWD$,1)="A" THEN 810
BO 660 IF LEFT$(FIRSTWD$,1)="P" THEN 790
HN 670 IF LEFT$(FIRSTWD$,1)<"0" OR LEFT$(FIRSTWD$
       ,1)>"9" THEN 1390
DJ 680 SECS=VAL(FIRSTWD$)
IL 690 '
NN 700 GOSUB 1140 '  FIRSTWD$ is "A.M." or "P.M."
II 710 IF FIRSTWD$="" THEN 760
HF 720 IF LEFT$(FIRSTWD$,1)="A" THEN 810
BJ 730 IF LEFT$(FIRSTWD$,1)="P" THEN 790
AH 740 GOTO 1390
HE 750 '
KG 760 IF HRS>12 THEN 810
```

```
BI 770 PRINT "A.M. or P.M. ?  "; : LINE INPUT INP
       TLN$ : GOSUB 1030
FO 780 GOTO 700
NI 790 HRS=HRS+12 ' p.m.
HL 800 '
NF 810 TM$=MID$(STR$(HRS),2)+":"+MID$(STR$(MINS),
       2)+":"+MID$(STR$(SECS),2)
HP 820 '
HI 830 ON ERROR GOTO 0  ' disable special error h
       andling
HD 840 '
FG 850 DATE$=DT$
AD 860 TIME$=TM$
IJ 870 '
NB 880 REM --- Turn off NUM LOCK mode
JP 890 POKE &H417,(PEEK(&H417) AND &HDF)
HM 900 '
CD 910 REM --- Exit to operating system (leave BA
       SIC)
JM 920 PRINT
MI 930 SYSTEM
MK 940 END
HG 950 '
II 960 '
GA 970 REM  -----------  SUBROUTINES  -----------
       -
MG 980 REM --- remove leading blanks from INPTLN$
NJ 990 WHILE LEN(INPTLN$)>0 AND LEFT$(INPTLN$,1)=
       " "
GO 1000    INPTLN$=MID$(INPTLN$,2)
EF 1010 WEND
IB 1020 RETURN
OL 1030 REM --- within INPTLN$, translate lower c
        ase to upper case,
JK 1040 REM     nonalphanumerics to blanks
JM 1050 C$=""
OP 1060 FOR I=1 TO LEN(INPTLN$)
DH 1070 CH$=MID$(INPTLN$,I,1)
EF 1080 IF (CH$>="A" AND CH$<="Z") OR (CH$>="0" A
        ND CH$<="9") THEN C$=C$+CH$ : GOTO 1110
JD 1090 IF (CH$>="a" AND CH$<="z") THEN C$=C$+CHR
        $(ASC(CH$)-32)  : GOTO 1110
LJ 1100 C$=C$+" "
FE 1110 NEXT I
LG 1120 INPTLN$=C$
IG 1130 RETURN
PB 1140 REM --- split INPTLN$ into FIRSTWD$ (firs
        t contiguous nonblank sequence)
DI 1150 REM     and INPTLN$ (the remainder)
EP 1160 GOSUB 980   ' remove leading blanks
```

```
DD 1170 FIRSTWD$=""
GI 1180 WHILE LEN(INPTLN$)>0 AND LEFT$(INPTLN$,1)
        <>" "
PK 1190   FIRSTWD$=FIRSTWD$+LEFT$(INPTLN$,1)
GC 1200   INPTLN$=MID$(INPTLN$,2)
FJ 1210 WEND
IF 1220 RETURN
EM 1230 REM --- handle errors in processing date
GN 1240 PRINT
ED 1250 PRINT "Please use one of the following fo
        rmats:"
AD 1260 PRINT
KF 1270 PRINT "    12 31 81                 12 31 1
        981"
BO 1280 PRINT "    12-31-81                 12-31-1
        981"
JP 1290 PRINT "    12.31.81                 12.31.1
        981"
PE 1300 PRINT "    12/31/81                 12/31/1
        981"
JF 1310 PRINT "    December 31, 1981        Dec. 31
        , 1981"
GJ 1320 PRINT
DF 1330 PRINT "(NUM LOCK is automatically on duri
        ng this routine"
OH 1340 PRINT "and off afterward; you can use the
         numeric keypad.)"
AC 1350 PRINT
AF 1360 PRINT
HA 1370 GOTO 110  ' retry
JJ 1380 '
BE 1390 REM --- handle errors in processing time
PF 1400 PRINT
EL 1410 PRINT "Please use one of the following fo
        rmats:"
QL 1420 PRINT
HP 1430 PRINT "    6 45 p.m.     6 45 p m      6 4
        5 p      18 45"
EG 1440 PRINT "    6:45 p.m.     6.45 p.m.     6:4
        5 p      18:45"
LB 1450 PRINT "    6:45:00 p.m.  6.45.00 p.m.  6:4
        5:00 p   18:45:00"
AH 1460 PRINT
ED 1470 PRINT "(NUM LOCK is automatically on duri
        ng this routine"
PF 1480 PRINT "and off afterward; you can use the
         numeric keypad.)"
BA 1490 PRINT
PH 1500 PRINT
HO 1510 GOTO 470 ' retry
```

Choosing a Display for Your PC/PCjr

Ottis R. Cowper

What's the most important interface in your computer system—the printer interface? The disk drive controller? Both may be indispensable, but think about the human interfaces which put you in contact with the computer—especially the monitor screen. If it's inadequate, eyestrain and headaches may be your fate. Here are some tips on choosing the best display for your needs.

The personal computer is a powerful tool for manipulating all sorts of information. However, the information inside a computer isn't useful unless it can be interpreted. Microcomputers communicate with their users via video displays, and IBM PC and PCjr users have a wide variety of display options from which to choose.

Before describing all these options, let's review the principles of video displays. It's easier to balance the various choices against each other if you understand, say, how an RGB monitor differs from a composite monitor.

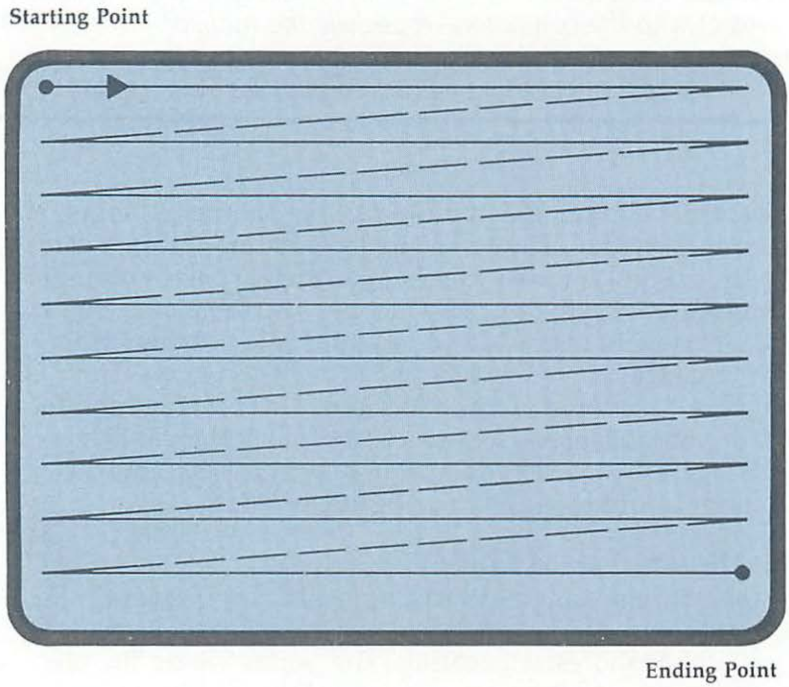
Monochrome Versus Color

For monochrome (single-color) displays, a hot filament called an electron gun sprays a beam of electrons onto a screen with a special phosphorescent coating. The points where the electrons strike the phosphor show up as glowing dots. Using electromagnets, the beam can be carefully targeted to determine exactly which dots will be lit, forming recognizable images.

Starting at the upper-left corner, the beam sweeps horizontally across to the right edge of the screen (see figure). It is then turned off and returned to the left edge during what is called the horizontal blanking period. The beam then sweeps across another line just below the first, and the process is repeated until the beam reaches the lower-right corner of the screen. At that point, the beam is turned off and returned to the top-left corner during what is called the vertical blanking period.

To control this process we need a video signal to determine which dots will be lit, an intensity signal to control how brightly the dots will be lit, and horizontal and vertical synchronization signals to control the blanking processes. These signals can be provided separately or combined into a *composite video* signal.

How Video Images Are Displayed



Starting at the upper-left corner of the screen, and moving left to right in this front view of the screen, the electron gun draws the scan lines (represented by solid lines). Then the gun shuts off for an instant (the horizontal blank period) as it returns to the left edge of the screen to draw the next scan line one step down. About 525 scan lines are standard on most TV sets and monitors. Finally, the gun shuts off again as it returns to the starting point. The process is repeated 60 times a second.

The procedure for creating color screen images is similar, except that three separate electron guns are used—one each for the primary colors red, green, and blue. All other colors are formed from combinations of these. That's why a multicolored display needs control signals for the red, green, and blue guns in addition to the intensity, horizontal, and vertical signals. As with monochrome displays, these signals can be provided separately or blended into a single composite video signal. As a general rule, the highest-resolution displays use the separate signals while the less expensive moderate-resolution models use composite video.

To determine which type of display you should choose, you'll need to consider how you'll be using your computer. For word processing and other applications involving only text, a monochrome monitor should be sufficient. Most people find monochrome easier on the eyes over long periods of viewing. On the other hand, if your applications require graphics you may need the color display.

PC Display Options

The choice is more critical for owners of the PC than the PCjr. The PCjr has built-in support for a variety of displays, while the PC has no built-in display capabilities.

IBM makes two adapter boards for the PC—the Monochrome Display and Printer Adapter and the Color/Graphics Monitor Adapter. The Monochrome Display Adapter provides video, intensity, and horizontal and vertical sync signals for the IBM Monochrome Display. This green-phosphor monitor provides sharp 80-column text displays, but the adapter doesn't support any type of graphics other than the graphics characters which are part of the standard PC character set. The Color/Graphics Monitor Adapter provides the necessary signals for the IBM Color Display, a high-resolution color monitor with inputs for direct red-green-blue (RGB) drive. The Color/Graphics Adapter also supports several different multicolor graphics modes.

Both adapters and their corresponding displays can be installed in the PC at the same time. Either may be specified as the default output device, but only one of the displays can be active at once. You can switch from one screen to the other using the MODE command from DOS. (See the IBM DOS

manual for more details.) Appendix I of the IBM BASIC manual shows how to switch displays from BASIC. Note that the IBM Monochrome Display cannot be used with the Color/Graphics Monitor Adapter and the IBM Color Display cannot be used with the Monochrome Display Adapter.

Ironically, if you want to use an inexpensive non-IBM monochrome monitor, you'll probably need to buy the Color/Graphics Monitor Adapter rather than the monochrome board. Few monochrome monitors use the separate signals provided by the Monochrome Display Adapter. Instead, they require composite video input, which is available from the color/graphics board in addition to the RGB direct-drive signals. A monochrome monitor connected to the color/graphics board's composite video plug can also display graphics. The different colors in the graphics display will appear as different brightnesses of the screen color.

You can also use the Color/Graphics Adapter's composite video output to connect a medium-resolution color monitor, which usually requires composite video. Such monitors are significantly less expensive than the high-resolution RGB displays—about \$250–\$400 versus \$680 for the IBM Color Display. However, 80-column text may be difficult to read, and the highest-resolution graphics modes may appear blurred. Only the RGB direct-drive type of monitor will have sufficient resolution to handle these displays in color. The IBM Color Display and similar high-res color monitors provide an 80-column text display which is only slightly less sharp than a monochrome display.

If you want to connect a non-IBM RGB color monitor to your PC, see the Option Instructions for the color/graphics board in your PC *Guide to Operations*.

Is a TV Good Enough?

It's also possible to use an ordinary TV set as a color or monochrome display. The color/graphics board includes circuitry which allows you to connect a device called an RF (Radio Frequency) modulator. The RF modulator is the interface to the TV set. You can think of the modulator as a super-miniaturized TV broadcasting station—it takes the video output from the color/graphics board and converts it into a simulated broadcast signal which is fed into the TV antenna terminals. Thus, the TV thinks it is receiving an ordinary TV station.

IBM doesn't make a modulator for the PC, but they are available from other suppliers for under $50. While this would provide an inexpensive display, you should note that 80-column text will be essentially impossible to read, and high-res color graphics will be blurred. Also, IBM recommends that you keep the TV at least 12 inches away from the disk drives; this will prevent any damage to disks from the strong electromagnetic fields created by televisions.

Of course, you're not compelled to buy IBM hardware. In the IBM market, there are almost always several other sources for any device you wish to add. A number of companies make display adapters with more features than the IBM boards. For example, many of the third-party adapters support both monochrome and color displays on the same board. Some of the color adapters can produce even higher-resolution displays, such as 640 × 400 dots in 16 colors, or even 720 × 348 dots. At least one of the monochrome boards extends the text capabilities to 132 columns per line. The problem with these enhanced displays is that they may not be compatible with your existing software. It would be wise to check before you purchase.

More Connectors on PCjr

With the PCjr there is no need to worry about adapters since the display circuitry is included on the computer's main circuit board. The PCjr has three separate video outputs. The connector labeled D (direct-drive) on the back of the System Unit provides all the signals for high-res RGB monitors. The jack labeled C (composite video) provides the signal for monochrome or medium-resolution color monitors. There's also a connector for an RF modulator (labeled T for television). IBM does make an RF modulator for use with the PCjr. (The warning about keeping the television at least 12 inches away from the disk drive also applies to the PCjr—resist the temptation to rest the TV atop the System Unit.)

All the same considerations which apply when choosing a monitor for the PC also go for the PCjr. If you have the standard 64K of memory (Entry Model PCjr), a medium-resolution composite video color monitor or TV will be sufficient for your display needs. If you have an Enhanced Model PCjr or if you add the 64K Memory and Display Expansion Option to the Entry Model, you gain the capability for 80-column text and

high-res color graphics (up to 640 $\times$ 200 dots in four colors). To see these clearly, you'll need either a composite video monochrome monitor or a high-res RGB monitor, depending on whether or not color is important to you. With the monochrome monitor, the different colors show up as different shades of the screen color. (Note that the IBM Monochrome Display cannot be used with the PCjr.)

One thing to keep in mind is that the Junior has an audio output for its sound chip, but no built-in speaker of its own. Some monitors contain audio amplifiers and speakers, and this would save you from adding these components separately if you want to make use of this special feature. The IBM Color Display has no audio capabilities.

Selecting the proper display is not an easy task. You'll have to make your decision after balancing the relative costs of the different options against the quality of screen images you need for your applications. Just as you shouldn't buy a stereo without a demonstration of how it sounds, so you shouldn't buy any display without a demonstration of its video quality. The ultimate criterion is whether it's a screen you'll be happy to stare into during the many hours you'll be spending with your computer.

Getting Started with a Disk Drive

Patrick Parrish

If you've just started using an IBM PC or PCjr and are new to PC-DOS (the Disk Operating System), here's a guide to the most common disk commands you'll need. It may save you some time wading through the IBM DOS manual.

The disk operating system for your IBM PC or PCjr is generally known as PC-DOS, or generically as MS-DOS, since it was written by Microsoft. PC-DOS offers a number of powerful commands for handling program and data files. In this article, we'll discuss how to use the DOS commands you'll need most often—including DIR, DISKCOPY, DISKCOMP, FORMAT, COPY, RENAME, and ERASE.

Since the PC was introduced in 1981, IBM has upgraded PC-DOS three times in an effort to keep it current. The various versions are numbered 1.00, 1.10, 2.00, and 2.10. The PCjr requires the latest version. Although the standard PC comes equipped with a single double-sided disk drive, many (if not most) PCs are dual-drive systems. For this reason, all versions of DOS have been written with two disk drives in mind, labeled drives A and B. At times this may lead to some confusion, especially on single-drive systems (such as the PCjr). If you have a single-drive machine, DOS will treat the drive as if it were both drives A and B. Fortunately, the screen instructions which appear during the execution of DOS commands explicitly state which drive is about to be accessed.

In order to use DOS, you must first *boot*, or load, DOS from the system master disk which accompanies your DOS manual. There are two standard ways to get DOS up and running. Simply place the DOS disk in your disk drive (drive A, if you have a PC with two drives) and switch on the computer. After a brief delay for a memory check, DOS is automatically booted.

The second method requires that the computer already be switched on. Sometimes this is referred to as *rebooting*. Place

the DOS disk in the drive (again, drive A with a dual-drive system) and simultaneously press the Ctrl (Control), Alt (Alternate), and Del (Delete) keys.

In either case, the first thing you'll see on the screen is a prompt to enter the date. Use this form: 4-15-1984 (month-day-year). After pressing the Enter key, another prompt asks for the time of day. Use 24-hour (military) time, such as 14:30 for 2:30 p.m. When you press the Enter key again, you'll see the standard DOS prompt:

A>

If you want, you can skip the date and time steps by just pressing the Enter key at each prompt. The advantage of answering the prompts is that whenever you save a file to disk, it will be "stamped" with the current date/time. This record appears when you call up a directory of the disk, and it's a good way to keep track of when a certain file was saved. That way, if you have several similar files on a disk, you can always be sure which one is the latest version you worked on.

Before describing how the DOS commands work, let's distinguish between two general types of commands—internal and external. The difference is important, because it determines whether or not the DOS disk must be inserted in the disk drive in order for the command to work.

Once you've booted DOS, the *internal* DOS commands are immediately available. They are called "internal" because the subprograms which handle these commands are loaded directly into the computer's memory. They remain there as long as DOS is present. The idea was to keep the most commonly used commands as accessible as possible.

On the other hand, *external* DOS commands are those which IBM decided would not be used as often. It would waste memory to keep the subprograms which handle these commands in the computer all the time. Instead, the commands are loaded into memory only when you use them. Since these commands are kept on the DOS disk, the disk must be inserted in the drive (drive A in dual-drive systems) whenever you use an external command. If you try to use one without the DOS disk in the drive, the computer alerts you to the error of your ways.

Now let's review the most common internal and external commands in PC-DOS. It would be a good idea to boot up DOS right now so you can get a feel for how they work.

CHAPTER 1

What's on the Disk

Probably the most frequently used DOS command is the one which lets you find out what files are stored on a disk. This command, DIR (short for DIRectory), is internal and thus can be accessed without the DOS system disk in the drive (provided, of course, that DOS has been booted).

When you type DIR and press Enter at the DOS prompt A>, the disk drive whirs for a second and the red LED lights up to indicate the drive is active. Then the screen lists all the files on the disk inserted in disk drive A. In addition to listing the filenames, DIR also shows you the length of each file (expressed in bytes, roughly equivalent to characters); the date and time the file was saved (assuming this information was updated when DOS was booted on that day); the number of files on the disk; and the amount of free space remaining on the disk (in bytes).

All of this information is given only for the disk in drive A (the lefthand drive on dual-drive systems). Drive A is always the *default drive*—the one referred to by the DOS prompt. This means that any DOS commands entered at this prompt will be executed on drive A.

At times, it may be more convenient to define drive B as the default drive. To do this, type B: and press the Enter key. The DOS prompt will change to B>. Now if you type the DIR command, the screen will list the files on the disk inserted in drive B. To restore the default to drive A, type A: and press Enter. This exchange works on single-drive systems, too; the lone drive can be referenced as either drive A or drive B.

However, there is an easier way to make a DOS command work on drive B instead of drive A. Simply append a B: when typing the DOS command at the A> prompt. For example, to list the files on the disk in drive B, type

DIR B: [and press Enter]

This same principle applies to all DOS commands. To divert the command to the drive opposite the one indicated by the prompt, just append the drive designation onto the end of the command.

Duplicating a Disk

As you become more adept with your PC or PCjr, undoubtedly you will find yourself using the DOS system disk

with greater frequency. It would be wise to make a backup copy of this disk as soon as possible. By using the backup as your system disk, you protect the original master disk from damage. Fortunately, there is an external DOS command which lets you easily back up the original DOS disk, or any disk which is not copy-protected (commercial software is often protected). The backup command is DISKCOPY.

The external DOS command DISKCOPY copies the contents of an entire disk. This is where a dual-drive system is really convenient. The disk you are copying (the *source disk*) goes into one drive, and the disk you're copying to (the *target disk*) goes into the other. The computer can easily transfer the contents of the source disk to the target disk. But on single-drive systems, only one disk can be inserted at a time. If the computer contained enough memory, it could simply read the entire contents of the source disk, wait for you to remove the source disk and insert the target disk, and then copy the files onto the target. However, there is rarely enough memory in the computer to hold the entire contents of the source disk. As a result, you'll have to repeatedly swap the source and target disks when using DISKCOPY on a single-drive system.

In the disk-copying routine with one disk drive, information from the source disk is first loaded into the computer's memory until the memory is filled. Then, DOS displays a message on the screen telling you to swap disks. After you insert the target disk and press any key, the information is copied onto the target. Next, DOS displays a message telling you to insert the source disk again. This process repeats until the entire disk is copied. The number of swaps depends on the amount of information on the source disk and the amount of memory in your computer.

On dual-drive systems, the transfer process is much simpler. Just insert the source disk (in this case, the DOS master disk) in drive A and the target disk (a blank disk) in drive B. Type the command

DISKCOPY [press Enter]

and follow the explicit instructions which appear on the screen. If the blank target disk was not previously formatted, DISKCOPY formats it for you. (We'll cover formatting in a moment.)

Although we chose to copy the DOS system disk from drive A to drive B, we just as easily could have copied from

drive B to A. To do this, first switch the default drive to drive B as discussed above (type B: and press Enter). Then, with the source disk in drive B and the target disk in drive A, type

DISKCOPY B: A: [press Enter]

How can you be sure that your source disk was copied successfully? We can verify that the information contained on the source and target disks is identical with another external DOS command, DISKCOMP.

In this procedure, the DOS messages on the screen refer to the source disk as the "first" disk and the target disk as the "second" disk. Otherwise, the DISKCOMP procedure is similar to using the DISKCOPY command. With single-drive systems, you alternate the source and target disks in drive A as directed. On dual-drive systems, the source disk is inserted into drive A and the target disk in drive B.

Assuming that you've successfully made a backup of the DOS system disk, we can now look at some other DOS commands. Be sure to put your original DOS master disk in a safe place and keep the copy for daily use.

Formatting

Another frequently used external DOS command is FORMAT. This command prepares a new blank disk for storing information. Every disk must be formatted before it is used for the first time.

FORMAT sets up a directory on the disk like the one we accessed earlier with the DIR command. Before formatting, make sure there is nothing of value to you on the disk. Formatting erases everything previously stored on the disk, and there's no way to recover the information.

The FORMAT command has several options. One option allows you to copy DOS files (called IBMBIOS, IBMDOS, and COMMAND) which contain the internal DOS commands. If you choose this option, you can boot DOS directly from your work disk without having to use the separate DOS disk. Of course, to use any external DOS commands, you still need the DOS disk.

Other FORMAT options let you assign a short name to the disk, limit the formatting to a single side of a disk (valuable if the disk will be used on both single- and dual-drive systems), and format a disk for use with older versions of

DOS. However, we won't cover these options in this article. Refer to your IBM DOS manual.

Now let's format a new bootable DOS disk. You'll need another fresh disk and a disk containing DOS (be sure the fresh disk has nothing important on it; FORMAT will erase any data that might be on the disk). Place the DOS disk in the drive (drive A with a dual-drive system) and type

FORMAT/S [press Enter]

The /S tells the computer you wish to format a disk with a copy of the DOS internal commands. Other options, if you choose them, can be tacked onto this command.

If you have a single-drive machine, start by inserting your DOS disk in the drive (remember, FORMAT is an external command which must be called off the DOS disk). The computer first reads the DOS internal command files into memory. As usual, the red LED glows while the drive is active. When the LED goes off, a screen message directs you to remove the DOS system disk from the drive and replace it with your new disk. Press any key and your fresh disk will be formatted.

To format a fresh disk on a dual-drive machine, place the DOS system disk in drive A and a new disk in drive B and type

FORMAT B:/S [press Enter]

When formatting is complete, you will be asked if you wish to format another disk. For now, type N (for No) so we can continue. Then enter DIR, and the directory for the new disk is displayed. Only one file should show up on the directory—COMMAND, one of the DOS files. Two more DOS files are actually on the disk (IBMBIOS and IBMDOS), but they don't show up on the directory. They are called "hidden" files.

Copying Files

Now that the work disk is prepared, we're ready to examine some DOS commands for manipulating individual files. These are all internal commands—COPY, RENAME, and ERASE (or DEL).

Let's begin by transferring a file from the DOS disk to the newly formatted work disk. Place your backup DOS disk in the drive (drive A with a dual-drive system) and enter DIR. The list of files is displayed on the screen. We can arbitrarily pick one of these files and copy it to the work disk. One such

file is CHKDSK.COM (.COM, for "command," is an *extension*, or modifier, that tells us the file type—in this case, an external command file). CHKDSK.COM checks a disk and reports certain information, such as how many files are listed in the directory, how many hidden files are on the disk, and so on.

For this exercise, the DOS disk is your source disk. To copy CHKDSK.COM from the DOS disk to your work disk on a single-drive system, make sure the DOS disk is still in the drive, and type

COPY CHKDSK.COM [press Enter]

DOS reads the file off the source disk into the computer's memory. Then a screen message tells you to insert the target disk (the work disk we created). After inserting the target disk, press any key. Because CHKDSK.COM is a short file, it can be copied in one pass. A longer file might require you to swap the source and target disks a few times, just as with the DISKCOPY command.

If you have a dual-drive system, place the source disk in drive A and the target disk in drive B. Enter the COPY command as listed above. The file will be copied directly from one drive to the other with no swapping required.

When the copy process is complete, check the directory with DIR to see that CHKDSK.COM is on the work disk (remember to use DIR B: to list a directory for a disk in drive B).

A New Name

Now suppose, after placing a file on a disk, you wish to change its name. To do this, you would use the RENAME (optionally abbreviated as REN) command. Let's change the name of the file CHKDSK.COM on the work disk (not your main DOS disk) to TEST.COM. First, place the work disk in the drive (drive A on a dual-drive system). Then type

RENAME CHKDSK.COM TEST.COM [press Enter]

Notice the form of this command. After RENAME, you specify the filename as it is presently listed in the directory followed by the new filename. The two filenames are separated by a space.

Again, we can check the disk directory to see that the program has been successfully renamed. Enter DIR and you should see that the program TEST.COM has replaced CHKDSK.COM in the directory.

Erasing a File

The last DOS command we'll consider is the ERASE, or DEL (Delete), command. This command lets you remove unwanted files from the disk. Use it carefully so that you don't unintentionally erase a file you really meant to keep.

ERASE (or DEL) is very straightforward. As an exercise, let's remove the file TEST.COM from our work disk. Type

DEL TEST.COM [press Enter]

and then examine the directory to see that only TEST.COM has been removed from the disk. The only remaining file should be COMMAND.

After trying these exercises and gaining some familiarity with these common DOS commands, try a few file manipulations on your own. Remember to keep the internal and external commands straight—external commands require the DOS disk to be in the drive. Luckily, DOS is fairly error-proof, and you'll be warned by screen messages if you make any mistakes. With a little practical experience, you will find that DOS commands are easy to work with and offer many conveniences for file manipulation on your PC or PCjr.

Super Directory

Charles Brannon

"Super Directory" is a black-and-white, 80-column directory program that will run on both a PCjr and a PC with or without the color/graphics adapter. It will help you keep track of your disks by allowing disk titles and 61-character extended filename descriptions. It runs on any IBM PC with DOS 2.0 or 2.1, or any Enhanced Model PCjr with Cartridge BASIC.

If you're not a programmer, you may find working with your PC or PCjr to be a bit perplexing at times. You switch on your system, wait as long as 45 seconds for the memory check, face the nondescript A> prompt in DOS, enter BASIC, load your program, and then finally run it. Things would be much easier if you had a list of all the programs and could run any one by just pressing a special function key.

"Super Directory" is the solution. When you run it, it displays a list of all the files on your disk. To run a program, just press the appropriate function key and the Enter key. Super Directory loads and runs the program for you.

Super Directory displays ten files on the screen at a time, one for each function key. If there are more than ten files on the disk, the list will be divided into several "pages" of screens. To turn the pages, press the Pg Dn or Pg Up key (found on the PC's numeric keypad, or accessed with the Fn and cursor keys on the PCjr). You can page forward or backward with Pg Dn and Pg Up.

There's more to Super Directory, too. PC-DOS limits you to 11-character filenames (8 characters plus a 3-character extender). Eleven characters don't allow very descriptive names for your files. How can you make sense of names like QTESTV1.BAS? But with Super Directory, you can append a description up to 61 characters long to each filename. The description is displayed every time you run Super Directory. You can also add titles to your disks which are displayed with the filenames and descriptions.

Creating Descriptions

After typing in Super Directory, run it. First, it asks you to select drive A: or B:. This is for dual-drive systems; if you have a PCjr or a PC with only one disk drive, type A. (You can also modify Super Directory for single-drive systems; see below.) The disk directory appears in a few seconds.

To enter a description, press the appropriate function key for the filename you wish to describe, and then press the space bar. The bottom line of the screen always prompts you what to do. The first time you attempt to enter a description, Super Directory asks for a disk name. Once you've entered this title, Super Directory remembers it and never asks you again.

Now you can enter the filename description. You don't have to use all 61 characters, of course. Press the Enter key when you're done.

After you enter a description, the screen is redrawn and you'll see the description next to the filename. If you want, you can repeat the process to change a description. The old description will be displayed at the bottom of the screen so that you can edit it. Remember to move the cursor to the end of the line when you're finished editing before pressing Enter. (You can press the End key to skip directly to the end of the line.)

When you run a program from Super Directory, the descriptions will be written to disk before the program is loaded. If you just want to save the descriptions without running a program, you can press the Esc key from the main menu. Another menu appears at the bottom of the screen:

1. Exit to BASIC
2. Exit to DOS
3. Re-Run
4. Save Descriptions
5. Menu

Press 4 to save the descriptions to disk. The descriptions are saved under the filename DESCR.DIR. Don't erase or delete this file or you'll lose your descriptions. The other options return you to BASIC or DOS, or rerun Super Directory. Pressing 5 returns you to the main menu (in case you pressed Esc by mistake).

A Few Hints

When Super Directory detects an error, it prompts you to press Enter. You'll usually be returned to the main menu. Don't try to run a program which is not in BASIC, however. You'll probably get the message "Direct Statement in File" and find that Super Directory has disappeared.

You can make Super Directory completely automatic. Enter this one-line command to make BASIC and Super Directory come up automatically whenever you switch on your computer (if you purchased the *COMPUTE!'s First Book of IBM* disk, an AUTOEXEC.BAT file has already been created):

OPEN "AUTOEXEC.BAT" FOR OUTPUT AS #1:PRINT #1,"BASIC SUPERDIR":CLOSE #1

This assumes that you've saved Super Directory to the same disk with the command SAVE"SUPERDIR", and that BASIC is also on the disk.

Super Directory normally asks you from which drive you want to read the directory (A: or B:). For single-drive systems you can remove the keyword REM from line 160. Leave the rest of the line in place.

Now you can add a flexible, easy-to-use menu to all the disks in your library. Super Directory even makes them easy enough for a child to use (which may or may not be a desirable feature).

Super Directory

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
DE 100 'Super Directory for PC/PCjr
JD 110 'for monochrome or color adapter, 80 colum
       ns
EB 120 SCREEN 0,0,0:WIDTH 80:COLOR 7,0:CLS:DEFINT
        A-Z:KEY OFF:FOR I=1 TO 10:KEY I,"":NEXT
ME 130 CR$=CHR$(17)+CHR$(196)+CHR$(217)
KE 140 PRINT"Welcome to ";:COLOR 15:PRINT"Super D
       irectory":COLOR 7
OC 150 'Remove the word REM from following line f
       or automatic use with drive A
QJ 160 REM DRIVE$="A:":FSPEC$="A:*.*":GOTO 200
JN 170 PRINT:PRINT "Select Drive: (";:COLOR 16,15
       :PRINT"A B";:COLOR 7,0:PRINT CHR$(29);CHR$
       (29);"/";CHR$(28);")"
EP 180 DRIVE$=INKEY$+":":A=ASC(DRIVE$):IF (A OR 3
       2)<97 OR (A OR 32)>98 THEN 180
```

```
JB 190 DRIVE$=CHR$(A AND 223)+":":FSPEC$=DRIVE$+"
       *.*"
AC 200 GOSUB 5000:CLS:COLOR 23:PRINT"Reading desc
       ription file"
NP 210 DIM D$(ENTRIES):FOR I=0 TO ENTRIES:D$(I)=C
       HR$(9)+"--" :NEXT
NP 220 ON ERROR GOTO 310
PA 230 OPEN DRIVE$+"DESCR.DIR" FOR INPUT AS #1
MG 240 LINE INPUT #1,DISKNAME$:LINE INPUT#1,A$:NU
       MREC=VAL(A$)
IN 250 FOR ITEMS=0 TO NUMREC
CC 260   LINE INPUT #1,F$:LINE INPUT#1,D$
PO 270   FOR I=0 TO ENTRIES
CF 280     IF F$=F$(I) THEN D$(I)=D$
HB 290     NEXT:NEXT
DI 300 GOTO 320
CP 310 RESUME 320
NJ 320 CLOSE#1:ON ERROR GOTO 0
DH 330 PAGES=INT(ENTRIES/10)
NL 340 CURR=0
KO 350 START=CURR*10:FINISH=START+9:IF FINISH>ENT
       RIES THEN FINISH=ENTRIES
EP 360 CLS:COLOR 0,15:PRINT STRING$(80,32):LOCATE
        1,2:PRINT"Super Directory";TAB(70);"Drive
       ";DRIVE$:LOCATE 1,40-LEN(DISKNAME$)/2:PRI
       NT DISKNAME$:PRINT
GK 370 FOR I=START TO FINISH
AC 380 COLOR 0,15:PRINT "F";LEFT$(MID$(STR$(1+I-S
       TART),2)+" ",2);:COLOR 15,0:PRINT " ";F$(I
       );TAB(18);:COLOR 7:PRINT D$(I):PRINT STRIN
       G$(80,196);
OB 390 NEXT
OF 400 LOCATE 25,1:COLOR 15,0:PRINT"Press ";:COLO
       R 0,15:PRINT"F1";:COLOR 15,0:PRINT" to ";:
       COLOR 0,15:PRINT "F";MID$(STR$(1+FINISH-ST
       ART),2);:COLOR 15,0:PRINT" to select progr
       am.  Press PgUp or PgDn to page, ESC to qu
       it.";
JD 410 LOCATE 23,32:PRINT"Page #";CURR+1;"of";PAG
       ES+1
CK 420 A$=INKEY$:IF A$="" THEN 420
GN 430 IF A$<>CHR$(27) THEN 540
JK 440 LOCATE 25,1:PRINT SPACE$(79);:LOCATE 25,1:
       PRINT"  1. Exit to BASIC  2. Exit to DOS
       3. Re-RUN  4. Save descriptions  5. Menu";
BF 450 A$=INKEY$:IF A$<"1" OR A$>"5" THEN 450
DB 460 ON VAL(A$) GOTO 470,480,490,500:GOTO 350
MH 470 COLOR 7:CLS:END
MN 480 SYSTEM
HA 490 RUN
```

```
FG 5ØØ ON ERROR GOTO 51Ø:GOSUB 1ØØØ:GOTO 35Ø
FC 51Ø BEEP:LOCATE 25,1:PRINT SPACE$(79);:LOCATE
       25,1:COLOR 31:PRINT"Can't save description
       s.  ";:COLOR 7:PRINT"Press ";CR$;" to cont
       inue.";
EE 52Ø IF INKEY$<>CHR$(13) THEN 52Ø
FM 53Ø RESUME 35Ø
DM 54Ø IF A$=CHR$(Ø)+CHR$(81) THEN CURR=-(CURR+1)
       *(CURR<PAGES):GOTO 35Ø
DH 55Ø IF A$=CHR$(Ø)+CHR$(73) THEN CURR=CURR-1:CU
       RR=CURR-(PAGES+1)*(CURR<Ø):GOTO 35Ø
QK 56Ø A=ASC(MID$(A$+"Ø",2))-59:IF A<Ø OR A>FINIS
       H-START THEN BEEP:GOTO 42Ø
CH 57Ø LOCATE 25,1:PRINT SPACE$(79);:LOCATE 25,1:
       PRINT"Press ";CR$;" to run program, ESC to
        return to menu, SPACE to do description."
       ;
IC 58Ø LOCATE 3+A*2,5:COLOR 31:PRINT F$(START+A);
       :COLOR 15
QL 59Ø A$=INKEY$:IF A$<>CHR$(13) AND A$<>CHR$(27)
        AND A$<>CHR$(32) THEN 59Ø
HM 6ØØ IF A$=CHR$(27) THEN LOCATE 3+A*2,5:PRINT F
       $(START+A);:GOTO 4ØØ
IN 61Ø IF A$<>CHR$(32) THEN 67Ø
JF 62Ø IF DISKNAME$="" THEN LOCATE 25,1:PRINT SPA
       CE$(79);:LOCATE 25,1:LINE INPUT;"Enter nam
       e of disk : ";DISKNAME$ :GOTO 62Ø
BI 63Ø LOCATE 25,1:PRINT SPACE$(79);:LOCATE 25,1:
       Z=START+A:PRINT "Description :";D$(Z);:LOC
       ATE 25,15:LINE INPUT ;D$(Z):D$(Z)=LEFT$("
       "+D$(Z),62):GOTO 35Ø
KC 64Ø LOCATE 25,1:PRINT SPACE$(79);:BEEP:COLOR 3
       1:LOCATE 25,1:PRINT"Cannot save descriptio
       ns to disk.  ":COLOR 7:PRINT"Run program a
       nyway? (Y/N):";:COLOR 7
MA 65Ø A$=INKEY$:IF A$<>"y" AND A$<>"Y" AND A$<>"
       n" AND A$<>"N" THEN 65Ø
LI 66Ø IF A$="y" OR A$="Y" THEN RESUME 68Ø ELSE R
       ESUME 35Ø
EH 67Ø ON ERROR GOTO 64Ø:GOSUB 1ØØØ
EO 68Ø ON ERROR GOTO 69Ø:COLOR 7:CLS:RUN DRIVE$+F
       $(START+A)
DM 69Ø LOCATE 25,1:PRINT SPACE$(79):COLOR 23:BEEP
       :PRINT"Cannot run ";F$(A);".  ";:COLOR 7:P
       RINT"Press ";CR$;" to continue...";
EA 7ØØ IF INKEY$<>CHR$(13) THEN 7ØØ
FK 71Ø RESUME 35Ø
LE 72Ø END
AD 1ØØØ 'Save descriptions to disk
EK 1Ø1Ø OPEN DRIVE$+"DESCR.DIR" FOR OUTPUT AS #1
```

```
NB 1020 PRINT#1,DISKNAME$;CHR$(13);ENTRIES;CHR$(1
        3);
QN 1030 FOR I=0 TO ENTRIES:PRINT#1,F$(I);CHR$(13)
        ;D$(I);CHR$(13);:NEXT
CH 1040 CLOSE #1:ON ERROR GOTO 0:RETURN
IK 1050 '
AB 5000 'This subroutine reads disk directory int
        o a string array
PD 5010 'Enter with FSPEC$, the file spec for the
         FILES command
EM 5020 'Exits with array F$, and NUMFILES, the n
        umber of files
HN 5030 'uses a temporary array, TT$, which is ER
        ASEd after use
IL 5040 '
LE 5050 DEF SEG=0:WIDTH 80
JM 5060 HEAD=1050:TAIL=1052:BUFFER=1054
OA 5070 CLS:COLOR 23,0,0:PRINT"Reading disk direc
        tory"
HK 5080 COLOR 0:ON ERROR GOTO 5100
BK 5090 FILES FSPEC$:ON ERROR GOTO 0:GOTO 5110
CD 5100 BEEP:COLOR 31:CLS:PRINT"Cannot read direc
        tory":COLOR 7:ON ERROR GOTO 0:END
IM 5110 DIM TT$(24):LOCATE 3,1:COLOR 7:ROWS=0
MK 5120 'Put code for End, Enter into keyboard bu
        ffer:
EH 5130 POKE HEAD,30:POKE TAIL,34:POKE BUFFER,0:P
        OKE BUFFER+1,79:POKE BUFFER+2,13:POKE BUF
        FER+3,28
FK 5140 LINE INPUT TT$(ROWS)
EL 5150 IF TT$(ROWS)<>"" THEN ROWS=ROWS+1:GOTO 51
        30
EK 5160 IF NOT DIMMED THEN DIM F$(ROWS*4-1):DIMME
        D=1
KI 5170 ROWS=ROWS-1
KP 5180 FOR I=0 TO ROWS
LC 5190  FOR J=0 TO 3
FF 5200    T$=MID$(TT$(I),J*18+1,12)
PJ 5210    IF T$<>"" THEN F$(ENTRIES)=T$:ENTRIES=
        ENTRIES+1
NB 5220  NEXT J
GA 5230 NEXT I
OM 5240 ERASE TT$:ENTRIES=ENTRIES-1
OI 5250 DEF SEG:RETURN
```

TERMPlus: Upload and Download with Cartridge BASIC

Dale McBane and Jeff Hamdani

TERM *is a simple terminal program built into PCjr Cartridge BASIC.* TERM, *however, lacks the capability to transfer files over the phone lines. The following modifications transform* TERM *into "TERMPlus," adding the upload/download features crucial to any good telecomputing program. TERMPlus requires an Enhanced Model PCjr with Cartridge BASIC.*

With the IBM Internal Modem (or any compatible modem) and PCjr Cartridge BASIC, you have all you need for basic telecommunications. The cartridge contains an unpublicized bonus: a terminal emulator program named *TERM* which allows you to communicate over the phone lines with mainframes, commercial information services, other IBM personal computers, or any home computer.

To load *TERM*, insert Cartridge BASIC and your DOS disk and turn on the computer. Then, after entering BASIC, simply type TERM. The *TERM* program—itself written in BASIC—will load from the BASIC cartridge and run.

TERM is menu-driven and easy to use. Full instructions are in your BASIC manual. But *TERM* has one serious shortcoming: it has no upload or download capability. Any information you send or receive is lost as soon as it scrolls off the screen.

Fortunately, since *TERM* is written in BASIC, it can be easily modified to include these powerful capabilities. We call the result "TERMPlus." With uploading, you can send text files or programs via modem to distant computers. With downloading, you can receive text files and programs from other computers and from Bulletin Board Systems. In effect, you can transform the simple *TERM* program into full-fledged telecomputing software. Although it won't be as luxurious as

the better commercial programs, it's an excellent way to get started—and if you already own Cartridge BASIC, it's free.

We've designed TERMPlus so that you can even exchange files with many non-IBM computers. The programming staff at COMPUTE! frequently use TERMPlus to transfer programs among our IBM computers and Apples, Commodores, Ataris, and our minicomputer-based in-house typesetting system.

Modifying *TERM*

The program listing at the end of this article makes the necessary modifications to *TERM*. *The listing contains the modifications only; it is not the complete program.* As part of Cartridge BASIC, *TERM* is copyrighted by Microsoft, so we have published only the lines required to modify the program for uploading and downloading. To create TERMPlus, you will have to merge these lines with the *TERM* program and then save the modified program to disk.

There are two ways you can do this. Here's the first method:

1. Load and run *TERM* as described above.
2. Press the F2 function key (Fn-2) to exit the program.
3. Type LIST. You should see the *TERM* program listed on the screen.
4. Delete lines 70, 110, 2000, and 2010.
5. Type in the new program lines at the end of this article.
6. When you're sure the new program lines are entered correctly, save the modified program by typing SAVE "TERMPLUS" (or any filename you want, of course). For safety's sake, do this before running the program for the first time.

The second method allows you to type in the modifications using the "Automatic Proofreader" utility (Appendix B).

1. Type in the new program lines with the Automatic Proofreader, making sure the checksums match as you proceed.
2. Type SAVE"TERMMODS" to save the lines to disk. The Proofreader automatically saves files in ASCII form.
3. Exit the Proofreader by typing BASIC.
4. Load and run *TERM* as described above. Press F2 (Fn-2) to exit *TERM*.
5. Type LIST to be sure the *TERM* program is in memory. You should see it list on the screen.

6. Delete lines 70, 110, 2000, and 2010.
7. Now, use the MERGE command to combine the modifications with the original *TERM*. Type MERGE"TERMMODS" (or whatever filename you chose when you saved the file).
8. Before trying the program for the first time, save it by typing SAVE"TERMPLUS".

A New Main Menu

The main menu has been changed slightly from the original version of *TERM*. To alter the settings of menu entries, type the number of the entry you wish to change, a comma, and the new setting. For example, to set your terminal for half-duplex operation you would type 4,N. From the main menu, you can set transmission speed, word length, parity, and the duplex mode, or you can send commands to the modem.

Option 5 on the menu sends any modem command. For more information on the commands, see "Guide to Operations," *IBM PCjr Internal Modem Installation and Operating Instructions*, page 10, or look in the *Technical Reference Guide* beginning on page 3-40.

The IBM Internal Modem lets you dial a number up to 33 digits long, and it can automatically answer incoming calls. For instance, to dial the number 1-800-555-4567, you would select menu option 5 and type

5,DIAL 18005554567,R

The R is a retry command, instructing the modem to keep dialing the number if the line is busy.

To initiate the automatic answering feature, select item 5 and type

5,C *n*,A

This command statement will answer the telephone after *n* rings, where *n* ranges from hexadecimal 0 to hexadecimal F (decimal 15).

The other modem commands are entered in a similar way.

Upload/Download Options

The main change we made to the original *TERM* program, of course, was the addition of uploading and downloading capabilities. TERMPlus has a 24K buffer in RAM (Random Access Memory) for sending and receiving files. That means the

maximum length file you can send or receive is 24K, unless you split the file into pieces.

You may wonder why a PCjr with 128K of RAM has only a 24K buffer. This is because BASIC can address only 64K of memory at a time. This severely limits the amount of memory available for the buffer. Another factor that limits buffer space is the way the buffer stores characters in memory. Each character received from the modem is stored by its ASCII value as an element of a large integer array. This means each stored character takes up two bytes. If the buffer were POKEd directly into a safe place in RAM, the buffer space could easily be doubled. However, we chose safety over size. Very few of the programs we've transferred approach the limit of the buffer, and using the array for storage leaves the memory housekeeping to BASIC.

Begin your telecomputing session as usual by establishing contact with the remote computer. When you enter the conversation (telecommunications) mode from the main menu by pressing F1, TERMPlus prints a new line of menu selections at the bottom of the screen:

f1=Menu f2=Ext f3=Nul f4=Break f5=Open Buf f6=Save Buf f7=Load Buf f8=Trans Buf

The first four options are unchanged from *TERM:* F1 returns you to the main menu (thereby disconnecting the link); F2 disconnects, stops *TERM,* and exits to BASIC; F3 sends a null string over the phone line; and F4 sends a Break character (CTRL-C) over the phone line.

TERMPlus adds four more options: F5 opens the buffer for downloading the text you're receiving (or closes the buffer if it's already open); F6 flips to a screen which allows you to save the buffer contents to disk; F7 flips to a screen which lets you load the buffer with a file from disk; and F8 transmits (uploads) the buffer over the phone line.

Downloading a File

Let's say you want to download a file from another computer or simply capture the text you're receiving for future reference. After the link with the remote computer is established and TERMPlus is in conversation mode, press F5 (Fn-5). Everything now appearing on your screen is also being stored in the buffer. If the buffer already contains some text, the new information will be appended. When you've finished receiving the

file or the text, press F5 again to close the buffer. You can open and close the buffer as often as you like.

Next, to save the buffer's contents, insert a disk in the drive and press F6. The drive whirs and a disk directory appears on the screen. This tells you which files are already stored on the disk, and how much blank space remains. TERMPlus then asks

Filename? (no extender)

Type in any standard eight-character filename, but without a three-character extender. TERMPlus adds its own extender to the file—.BAS if it's a BASIC program, and .ASC if it's an ASCII text file. After you press the Enter key, TERMPlus asks

N)ew file or A)ppended file

If you want the buffer saved as a new file on your disk, press the N key. If you want the buffer appended onto an existing file, press A. Appending allows you to download files larger than 24K by receiving them in pieces. For example, if you're downloading a file that is 64K long, the buffer will fill with the first 24K. Then TERMPlus tells the transmitting computer to wait, informs you that the buffer is full, and tells you to save the buffer contents. After you finish saving, TERMPlus sends a CTRL-Q to signal the remote computer to resume sending the file. When the buffer fills the second time, save the contents by appending them to the first section, already on disk. So far you've received 48K. Then, when the remaining 16K section of the program is in the buffer, append the contents again. The 64K file will now be complete.

When appending a file, remember to specify the correct filename. Otherwise, the buffer contents will be tacked onto some other file on your disk.

Finally, after you answer New file or Appended file, TERMPlus asks

P)rogram file or T)ext file

If the file you downloaded is a BASIC program, press P. The file will be saved as a BASIC program file that can be loaded and run as usual. If you downloaded a text file, press T. The buffer will be saved on disk as an ASCII file that can be loaded with a text editor or word processor. (If you save a BASIC program as a text file and later attempt to run it, the computer locks up.)

As TERMPlus saves to disk, it prints the buffer contents on the screen. This lets you check for any garbage characters that may have garbled the file. Unfortunately, it also slows down the program. Because TERMPlus is written in BASIC, not machine language, it is slow compared with good commercial terminal programs. You can speed up the saving process a bit by removing the screen-print feature. See the section headed "Program Modifications" below.

After the buffer is saved, TERMPlus returns you to the conversation mode so that you can continue communicating. Saving erases the buffer, so you can reopen the buffer to download another file or capture more text.

Uploading a File

Let's say you want to transmit (upload) a file from your computer to the remote computer. After establishing the link and entering conversation mode, you need to load the file from your disk into the TERMPlus buffer. *The file on your disk must be in ASCII format.* If you want to upload a BASIC program, you'll have to convert it to ASCII first by saving it to disk with this command:

SAVE"filename",A

To load the file into the TERMPlus buffer, put the disk in the drive and press F7 (Load Buf). TERMPlus asks

The current contents of the buffer will be lost.
ARE YOU SURE?

If you don't want to erase the buffer—for example, if you pressed F7 by mistake—press N. TERMPlus returns you to the conversation mode. If you press Y, the buffer is cleared, a disk directory appears on the screen, and TERMPlus asks

Filename (include extender)?

Type in the entire name of the file you want to upload and press Enter. TERMPlus loads the file from disk and stores it in the buffer, then returns you to the conversation mode. As the file is loading into the buffer, you'll see it printing on the screen. Again, this gives you an opportunity to check for any garbage characters that may have crept in. (If you want to speed up this process by removing the screen-print feature, see the section on "Program Modifications.")

When you're ready to transmit the file, press F8 (Trans Buf). TERMPlus begins uploading and reports how many bytes have been sent. When the transfer is complete, TERMPlus returns you to the conversation mode.

Inside TERMPlus

To create TERMPlus, four new subroutines have been added to *TERM:*

Lines 7020–7070 are a toggle switch routine which opens and closes the buffer when F5 is pressed. The buffer is stored in the integer array BUF.

Lines 8000–8190 save the buffer. They read each byte from the 24K buffer and write it to the disk. When the buffer contains a BASIC program file, TERMPlus searches the buffer for the first line number and throws out any trash which may have preceded it. After finding the line number, the routine saves everything in the buffer on disk until it encounters a carriage return. Then it searches for another line number and repeats the process.

Lines 9000–9150 load the buffer from disk. The previous contents of the buffer are erased.

Lines 10000–10060 transmit the contents of the buffer via the modem to the remote computer.

Program Modifications

To stop TERMPlus from printing the buffer on the screen as it saves to disk, change line 8160 to read

8160 IF BYTE=13 THEN PRINT #3,CHR$(13);CHR$(10);:FLAG=0 :ELSE PRINT #3,CHR$(BYTE);

To stop TERMPlus from printing the buffer on the screen as it loads from disk, remove the following PRINT statement from line 9120:

:PRINT B$;

TERMMODS

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
IO 10 WIDTH 80:DEF SEG:DEFINT A-Z:BFSZ=24:REM siz
      e of buffer
IB 15 DIM BUF(BFSZ*1024)
DF 60 DATA "Full Duplex",8,23,"Y","(Y or N)"
MJ 80 DATA "Modem Command",10,18,"",""
ML 90 FOR I=1 TO 5:READ TXT$(I),ROW(I),COL(I),FD$
      (I),F$(I):NEXT I
```

```
HO 1ØØ XON$=CHR$(17):XOFF$=CHR$(19):Z=Ø:Z$="":POINTER!=Ø
HA 13Ø ON ERROR GOTO Ø:ID$="Terminal Emulator with Upload/Download"
JO 1Ø1Ø ON KEY(3) GOSUB 52ØØ:ON KEY(4) GOSUB 53ØØ:ON KEY(5) GOSUB 7Ø2Ø
NM 1Ø15 ON KEY(6) GOSUB 8ØØØ:ON KEY(7) GOSUB 9ØØØ:ON KEY(8) GOSUB 1ØØØØ
LM 1Ø2Ø FOR I=1 TO 2:KEY (I) ON:NEXT I:GOSUB 2Ø2Ø
EJ 1Ø3Ø LOCATE 18,1,1,Ø,7:PRINT"Change <line,data>?";STRING$(61," ")
OG 111Ø I=VAL(D$):IF (I<1) OR (I>5) THEN 1Ø3Ø
FN 112Ø FD$(I)=RIGHT$(D$,LEN(D$)-X):GOSUB 2Ø2Ø:GOTO 1Ø3Ø
MN 2Ø2Ø ON ERROR GOTO Ø:CLS:K$="f1=Conv":Z$="":GOSUB 6ØØØ:PRINT ID$:PRINT
DL 2Ø3Ø FOR I=1 TO 5
LO 3ØØØ DV$="1:":IF VAL(FD$(1))>3ØØ THEN DV$="2:"
LD 4Ø2Ø ON KEY(1) GOSUB 5ØØØ:ON KEY(2) GOSUB 51ØØ:KEY(3) ON:KEY(4) ON:KEY(5) ON:KEY(6) ON:KEY(7) ON:KEY(8) ON
IO 4Ø3Ø CLS:GOSUB 7ØØØ:K$="f1=Menu":Z$="f5=Open Buf f6=Save Buf f7=Load Buf f8=Trans Buf":GOSUB 6ØØØ
KO 4Ø5Ø IF A$<>CHR$(1Ø) THEN PRINT A$;:IF Z THEN BUF(POINTER!)=ASC(A$+CHR$(Ø)):POINTER! = POINTER! + 1
CG 4Ø51 IF POINTER!>BFSZ*1Ø24 THEN PRINT#1,XOFF$:PRINT "BUFFER FULL.  Press return to save buffer contents.":INPUT Q$:GOSUB 8ØØØ:GOSUB 7Ø2Ø :PRINT#1,XON$
EJ 5ØØØ COM(1)OFF:KEY(3)OFF:KEY(4)OFF:KEY(5)OFF:KEY(6)OFF:KEY(7)OFF:KEY(8)OFF:ON KEY(2) GOSUB 511Ø:CLOSE:CLS:RETURN 1ØØØ
KI 54ØØ IF LOC(1)>2ØØ THEN PRINT #1,XOFF$;:PAUSE=-1
HN 55ØØ IF LOC(1)<1Ø AND PAUSE THEN PRINT #1,XON$;:PAUSE=Ø
GA 6ØØØ LOCATE 25,1:PRINT K$+" f2=Ext f3=Nul f4=Break "+Z$;
CP 7ØØØ IF FD$(5)<>"" THEN PRINT #1,CHR$(14)+FD$(5)
FP 7Ø2Ø Z=(-1-Z)
KI 7Ø3Ø IF Z THEN GOTO 7Ø6Ø:REM if opening buffer
NG 7Ø4Ø Z$="f5=Open  Buf f6=Save Buf f7=Load Buf f8=Trans Buf":GOTO 7Ø7Ø
BI 7Ø6Ø Z$="f5=Close Buf f6=Save Buf f7=Load Buf f8=Trans Buf"
EB 7Ø7Ø CLS:LOCATE 25,1:PRINT K$+" f2=Ext f3=Nul f4=Break "+Z$:LOCATE 1,1:RETURN
```

```
KJ 8000 REM SAVE BUFFER ROUTINE
DI 8002 IF Z THEN GOSUB 7020: REM close buffer
HG 8005 CLS:PRINT"                    SAVE BUFFER":PRIN
        T:FILES:PRINT
MN 8010 ON ERROR GOTO 8010
CD 8020 PRINT "Filename?(no extender)";:NM$=""
HH 8030 Q$=INKEY$:IF Q$="" THEN 8030
NM 8040 IF ASC(Q$)=13 THEN 8070
MG 8045 IF ASC(Q$)=8 THEN PRINT CHR$(29);CHR$(32)
        ;CHR$(29);:NM$=LEFT$(NM$,LEN(NM$)-1):GOTO
         8030
HN 8050 IF ASC(Q$)<48 OR (ASC(Q$)>57 AND ASC(Q$)<
        65) OR (ASC(Q$)>90 AND ASC(Q$)<97) OR ASC
        (Q$)>122 THEN 8030
GN 8060 PRINT Q$;:NM$=NM$+Q$:GOTO 8030
DF 8070 IF LEN(NM$)>8 THEN NM$=LEFT$(NM$,8)
CB 8075 PRINT:PRINT:PRINT "N)ew file or A)ppended
         file"
IL 8080 Q$=INKEY$:IF Q$<>"n" AND Q$<>"N" AND Q$<>
        "a" AND Q$<>"A" THEN 8080
GA 8082 PRINT:PRINT:PRINT "P)rogram file or T)ext
         file"
CD 8084 Q$=INKEY$:IF Q$<>"p" AND Q$<>"P" AND Q$<>
        "t" AND Q$<>"T" THEN 8084
GL 8086 EX$=".bas":IF K$="t" OR K$="T" THEN EX$="
        .asc"
KI 8088 IF Q$="n" OR Q$="N" THEN 8100
NG 8090 FLAG=1:OPEN NM$ + EX$ FOR APPEND AS 3:ON
        ERROR GOTO 0:GOTO 8110
GB 8100 FLAG=0:OPEN NM$ + EX$ FOR OUTPUT AS 3:ON
        ERROR GOTO 0
FK 8102 PRINT:PRINT:PRINT "P)rogram file or T)ext
         file"
CC 8104 Q$=INKEY$:IF Q$<>"p" AND Q$<>"P" AND Q$<>
        "t" AND Q$<>"T" THEN 8104
BN 8110 POINTER2!=0:PRINT:PRINT POINTER!;"bytes"
LN 8120 WHILE POINTER2!<POINTER!
OA 8125 IF Q$="t" OR Q$="T" THEN FLAG=1
PH 8130 BYTE=BUF(POINTER2!)
EA 8140 IF (BYTE<48 OR BYTE>57) AND FLAG<>1 THEN
        GOTO 8170
FF 8150 FLAG=1
GM 8160 IF BYTE=13 THEN PRINT #3,CHR$(13);CHR$(10
        );:PRINT CHR$(13);CHR$(10);:FLAG=0:ELSE P
        RINT #3,CHR$(BYTE);:PRINT CHR$(BYTE);
OD 8170 POINTER2!=POINTER2!+1
GD 8180 WEND
QM 8190 CLOSE 3:POINTER!=0:DEF SEG:GOTO 7040
OF 9000 REM LOAD BUFFER ROUTINE
DJ 9002 IF Z THEN GOSUB 7020: REM close buffer
```

```
BC 9005 CLS:PRINT"                    LOAD BUFFER":PRIN
        T
IM 9010 PRINT"The current contents of the buffer
        will be lost."
GC 9020 PRINT"ARE YOU SURE?";
FI 9030 Q$=INKEY$:PRINT Q$;:IF Q$="" THEN 9030
KJ 9040 IF Q$<>"y" AND Q$<>"Y" THEN 7060
ME 9050 PRINT:FILES:PRINT
FP 9060 ON ERROR GOTO 9060
JP 9070 PRINT"Filename(include extender)";:INPUT
        NM$
QK 9080 IF LEN(NM$)>12 THEN GOTO 9070
PA 9090 OPEN NM$ FOR INPUT AS 3:ON ERROR GOTO 0
IE 9100 POINTER!=0
OP 9110 WHILE EOF(3)=0 AND POINTER!<=BFSZ*1024
DH 9120 B$=INPUT$(1,3):BUF(POINTER!)=ASC(B$+CHR$(
        0)):POINTER!=POINTER!+1:PRINT B$;
FF 9130 WEND
KL 9140 IF POINTER!>BFSZ*1024 THEN PRINT:PRINT "B
        UFFER FULL: remainder of file truncated!"
EI 9145 PRINT:PRINT POINTER!;"bytes":PRINT:PRINT
        "Press return to continue":INPUT Q$
PC 9150 CLOSE 3:DEF SEG:GOTO 7020
CH 10000 REM TRANSMIT BUFFER ROUTINE
DN 10002 IF Z THEN GOSUB 7020
NL 10005 CLS:PRINT"              TRANSMITTING BUFFER"
JM 10010 POINTER2!=0:PRINT:PRINT POINTER!;"bytes"
EB 10020 WHILE POINTER2!<POINTER!
DM 10030 BYTE$=CHR$(BUF(POINTER2!)):PRINT BYTE$;:
         PRINT #1,BYTE$;
GG 10040 POINTER2!=POINTER2!+1
JE 10050 WEND
NC 10060 Z = (-1-Z) : GOTO 7020
```

Disk Rx

Michael A. Covington

Did you erase a file by accident? Maybe you can resurrect it with "Disk Rx." This interesting utility also lets you explore how information is stored on your disk. It runs on any PC with at least 128K RAM and disk drive, or Enhanced PCjr with Cartridge BASIC. For DOS 2.0 or 2.1

If you're anything like me, about once a month the dreaded misfortune befalls you: You find you've erased your only copy of an important program or document. Perhaps you thought you had a second copy when you didn't, or perhaps it was just a silly mistake. Maybe you typed the wrong filename because you were thinking about two files at once.

Fortunately, deleted files can often be recovered. When you delete a file from an IBM PC or PCjr disk, the data isn't really erased. Instead, the space on the disk that the file occupied is marked as free space for future use. The contents of the file aren't overwritten until the space is needed for something else. So if you've accidentally deleted a file—and you haven't yet written anything else onto the disk—there's at least a chance you can get the file back. "Disk Rx" lets you search through a disk to locate your data, reconstruct the file in a memory buffer, and then save a copy on a new disk.

How to Recover a File

In brief, here are simple instructions for using Disk Rx:

1. Make sure you write nothing else on the disk containing the data you want to recover. Any new files or data would probably overwrite the deleted file. Immediately after the accident, remove the disk from the computer and set it aside.
2. Call up Disk Rx and start it running. When it's ready, insert the disk with the deleted file into the drive. By following the Disk Rx screen instructions, you can display sectors of the disk on the screen, one at a time.

3. Step through the disk, sector by sector, hunting for pieces of your lost file. Press N (for *Next unallocated sector*) as many times as necessary until you come to the first sector of your file. Then press the A key to add that sector to the Disk Rx memory buffer. Then press N again to find the next sector, and so forth. Nine times out of ten, you can work your way through the whole file using just the N and A keys; you'll know you're done when you reach a sector that isn't part of the file.
4. After you've put together all of your file in memory (the Disk Rx buffer holds up to 48K), press Q to quit. Disk Rx asks you to insert another disk and type a filename. Then it saves a copy of the rebuilt file.

But sometimes it's not this simple. It's not always easy to recognize pieces of your deleted file or to determine their proper order. Some skill is involved; Disk Rx can't work miracles by itself. As we'll explain in a moment, if you accidentally delete a file from a heavily used disk, the sectors you need to retrieve may be scattered all over the disk and are difficult to reorganize. Occasionally, it may be easier to recreate a lost file than to reconstruct it. That's a choice you'll have to make after examining a problem disk with Disk Rx.

Searching for Sectors

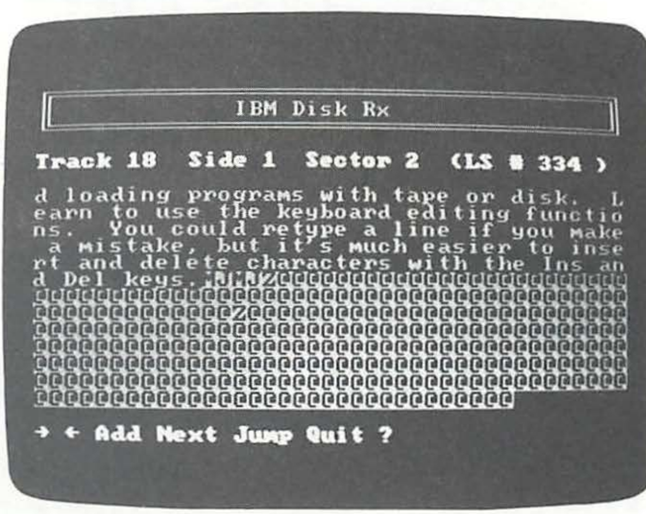
To use Disk Rx effectively, you must learn to recognize fragments of your deleted file when you see them. As you view the contents of each sector on the screen, keep in mind that Disk Rx has to translate some characters to make them displayable. For example, Disk Rx does not skip to a new line when it encounters a carriage return character in a sector. If it did, an entire sector might not fit on the screen. Instead, control characters such as carriage returns and linefeeds are represented by adding 32 to their ASCII values and displaying the result in reverse. This is like pressing a Ctrl key combination on the keyboard—a Ctrl-A, for instance, shows up as the letter A in reverse video.

If you accidentally erased a text file, your job is relatively easy. Most of the text *does* appear onscreen as normal characters. For ASCII text files, the end of a line is represented by Ctrl-M and Ctrl-J (carriage return and linefeed). The end of a

file is usually denoted by Ctrl-Z, after which there may be random data (usually repeated material from the previous sector). Most word processing files follow this system (Figure 1), though additional special characters may also be present.

Other types of files look quite different. Tokenized BASIC programs, for instance, are a jumble of characters in which only the variable names, quoted strings, and comments are recognizable (Figure 2). And in a machine language program, you may not be able to recognize anything except a few messages (Figure 3). The main challenge in these cases is to identify the beginning of your file. Also, these files usually don't end with Ctrl-Z like text files. However, spurious material at the end of a program is unlikely to cause problems if the program is a closed loop.

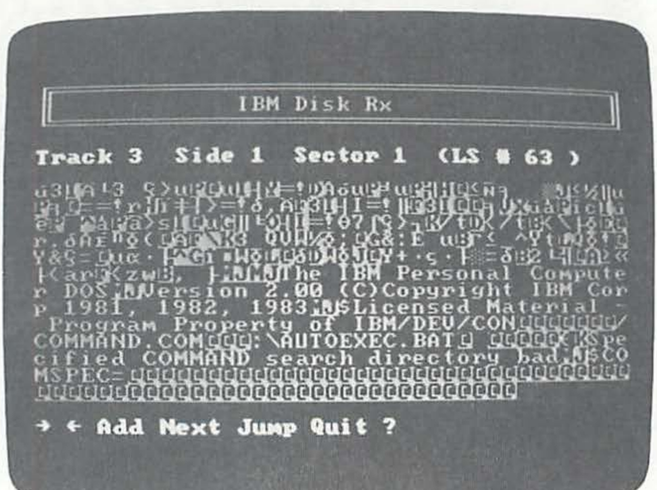
Figure 1. A sector containing a text file is easy to spot. Notice the Ctrl-M, Ctrl-J denoting the end of a line, and the Ctrl-Z at the end of the file.



To get a feel for how different types of files appear onscreen, practice examining some sectors with Disk Rx. The utility won't alter or harm your disks in any way.

Figure 2. Sectors containing tokenized BASIC programs are harder to identify. Only character strings, variable names, and comments are readable.

Figure 3. Sectors which store pieces of machine language programs are the hardest to recognize. Everything but the comments appears as garbage.



Restoring Order

Your second task when reconstructing a deleted file is putting the pieces back together in the proper sequence. This is made a lot harder by the way an IBM computer stores files on a disk.

The basic problem is that files aren't necessarily stored in blocks of contiguous sectors. For instance, assume that you've saved two files on a freshly formatted disk. Now let's say you delete the first file, then save a new file which is longer than the deleted one. DOS (the Disk Operating System) starts writing the new data in the gap left by the deleted file. When it runs out of room there, it skips over the sectors occupied by the second file and saves the rest of the new data in another block. Therefore, the new file is split into two parts, or noncontiguous blocks, around the existing file.

As you might expect, a heavily used disk—on which you've saved and deleted many files of different lengths—can get pretty messy. A long file might be scattered in separate blocks all over the disk. They're not only noncontiguous, but also nonconsecutive. The last part of a file might be stored in a block before the first part of a file.

All this is normally something you'd never think about because DOS takes care of the messy details. DOS keeps a map on the disk to keep track of which blocks of sectors belong to which files (called *allocation*). The map also tells DOS how to put the sectors together in the proper order (called *linkage*).

When you delete a file, the sectors of actual data are left undisturbed, but unfortunately, the *map* is altered. All of the allocation and linkage information is erased.

Some disk operating systems (for example, Commodore and Atari) store the linkage information within the data sectors themselves. It's therefore easy to reconstruct deleted files on these disks since each sector points to the next sector in the linked chain of files. But since PC-DOS stores the linkage information in a map which is altered when the file is deleted, you have to link sectors back together the hard way. If the disk hasn't been used much, you might be lucky enough to find all or most of the sectors contiguous and consecutive. Otherwise, you must examine each sector, one by one, and restore the proper order yourself. Fortunately, Disk Rx has a few features to make the job possible.

Solving the Puzzle

When sectors of a deleted file are randomly scattered all over a disk, your best bet is probably to recover your file in more than one piece, then put the pieces together in the correct order using the COPY command in DOS.

To do this, retrieve a block of sectors using the Disk Rx buffer. Save each block as a separate file on a scratch disk. When you've collected all the blocks that belong to the lost file, try to reassemble them on the scratch disk in the right order with COPY. (See your DOS manual for instructions on combining, or *concatenating*, multiple files with the COPY command.)

An alternative is to search through the disk until you find out where the scattered blocks of sectors are, then use the Disk Rx buffer to reassemble them in the proper sequence. Disk Rx lets you jump directly to any sector when you press the J key.

One warning is in order: If you are piecing together a file and cannot locate the original last sector of the file, you'll get strange results when you try to reload the partial file. Without the last sector, the file will be missing its end-of-file marker (the Ctrl-Z character mentioned above). In this case, random data will appear to be appended to the end of the file when it is reloaded into memory. Or if you have a program already in memory when you load the partial file, any portion of the program in memory that's beyond the end of your partial file will be added to the file.

Exploring Disks

You can also use Disk Rx to examine the contents of any sector on a disk, whether or not it is part of a file. You can even examine non-IBM disks, as long as they have 40 tracks and eight or nine 512-byte sectors per track. (This doesn't mean that the program can read *any* disk; generally only IBM-compatible computers use the disk format of 40 tracks with eight or nine sectors of 512 bytes.) Internally, DOS refers to sectors by their logical sector numbers (Table 1). The same numbering system is used for all disks, though some disks lack some of the sectors (there is no side 1 on a single-sided disk, and no sector 9 on any track of an eight-sector disk).

Table 2 is a guide to the special areas that DOS records at the beginning of each disk. The boot record is generated by the FORMAT command in order to distinguish between system and nonsystem disks. Then comes the *file allocation table* (FAT), the map which identifies the status of each sector (free space, last sector of a file, or part of a file which continues in sector number so-and-so). The FAT looks like gibberish on the screen; each entry occupies one and a half bytes, and fairly elaborate techniques are needed to decode it.

Table 1. Sector Numbering In DOS 2.0 and 2.1

The same numbering system is used for all disks, and DOS keeps tabs on which numbers are not used on particular disks (that is, sector 9 of any track does not exist on eight-sector disks, and side 1 does not exist on a one-sided disk).

Logical Sector Number	Track	Side	Sector
0	0	0	1
1	0	0	2
2	0	0	3
3	0	0	4
4	0	0	5
5	0	0	6
6	0	0	7
7	0	0	8
8	0	0	9
9	0	1	1
10	0	1	2
11	0	1	3
12	0	1	4
13	0	1	5
.	.	.	.
.	.	.	.
.	.	.	.
717	39	1	7
718	39	1	8
719	39	1	9

Table 2. Special Sectors at the Beginning of Disks

8-sector, 1-sided disk	8-sector, 2-sided disk	9-sector, 1-sided disk	9-sector, 2-sided disk	Area
0	0	0	0	Boot record
1	1	1–2	1–2	First copy of FAT
2	2	3–4	3–4	Second copy of FAT
3–6	3–7, 9–10	5–8	5–11	Directory

Finally, there is the directory (Figure 4), which lists all the files (and/or subdirectories) together with codes for the sectors in which they begin. Examining the directory can be a good way to find out what files have been erased, since the filenames are clearly legible and a special character replaces the first letter of the name of each erased file. Note also that the directory sectors list hidden files such as IBMBIO.COM, even though the DIR command doesn't.

Figure 4. This sector contains directory information for the disk.

How It Works

The heart of Disk Rx is a machine language routine that asks DOS to read a particular sector, identified only by logical sector number. The statements in lines 90–210 POKE this routine into memory. You can use this routine in your own programs to read disks sector by sector. Note that before the routine is called, DOS must be notified that the user may have changed disks; this is done by executing any DOS operation that refers to drive A. I used CHDIR (line 260) because it does not cause an error if a non-IBM disk is inserted.

To set aside a large buffer in which to store the reconstructed file, Disk Rx manages memory in a somewhat unusual way. The CLEAR statement in line 50 tells the BASIC interpreter not to use addresses from hexadecimal 3C00 to the top of the 64K segment. As indicated in Table 3, this area contains a 48K workspace in which the recovered file is put together, followed by the machine language subroutine and some space for its parameters, including the 512-byte sector itself.

To identify unallocated sectors (which are likely to contain deleted files), the program decodes the FAT entry for each sector and searches for values of 0. This is done in lines 1130–1170 and 1660–1720.

The rest of the program is relatively simple. As each sector is read, Disk Rx displays the contents by PEEKing the 512 bytes of memory beginning at hex FD00. If the sector is to be added to the workspace, these 512 bytes are POKEd into locations starting at STP (Storage Pointer), which is initially hex 3C00. At the end, the contents of the workspace are written out to a file one byte at a time.

Table 3. Special Disk Rx Memory Locations

(Hex)	
0000–3BFF	BASIC program and variables
3C00–FBFF	48K buffer for storing recovered files
FC00–FC20	Machine language subroutine to read disk sectors
FD00–FDFF	Transfer area for contents of sector
FF00	Indicates which disk drive to read (0=A, 1=B, etc.)
FF01	Low byte of logical sector number
FF02	High byte of logical sector number
FF03	Return code from subroutine (successful read=0, error=1)

Table 4. Important Disk Rx Variables

DISKREAD	Address of machine language subroutine
FAT	Array in which file allocation table is stored
FATP	Pointer used to read FAT
FATSIZE	Number of elements actually used by array FAT
FATV	Decoded FAT entry for a particular sector (unallocated sector=0)
NONDATA	Number of sectors occupied by boot record, root directory, and file allocation table. Counts the nonexistent ninth sector on 8-sector disks.
PCT!	Percentage of buffer used
SCTR	Logical sector number of current sector
SECTORS	Sectors per track (8 or 9)
SIDES	1-sided disk=1, 2-sided disk=2
STP	Points to next available byte in buffer where file is retrieved
STPF!	STP in floating-point form
SUM	Checksum to verify correct typing of DATA statements

Disk Rx

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
DC 10 REM IBM Disk Rx
EM 20 SCREEN 0: WIDTH 40: CLS: KEY OFF
DM 30 DEF SEG: POKE 91,1: POKE 92,25
PB 40 GOSUB 1500
EA 50 CLEAR,&HFF10: CLEAR,&H3C00
CP 60 DEFINT A-Z
JH 70 OPTION BASE 0: DIM FAT(536)
GJ 80 ' Poke mach.lg. routine into place
MP 90 I=&HFC00: SUM=0
EJ 100 READ J: IF J>255 THEN 120
FP 110 POKE I,J: SUM=SUM+J: I=I+1: GOTO 100
FD 120 IF SUM<>J THEN PRINT "Typing error in DATA
        statements":END
ID 130 DATA &HA0, &H00, &HFF, &H8B, &H16
FO 140 DATA &H01, &HFF, &HB9, &H01, &H00
NC 150 DATA &HBB, &H00, &HFD, &HCD, &H25
PO 160 DATA &H73, &H07, &HC6, &H06, &H03
IB 170 DATA &HFF, &H01, &HEB, &H05, &HC6
FA 180 DATA &H06, &H03, &HFF, &H00, &H9D
DH 190 DATA &HCA, &H00, &H00, 3346
DL 200 POKE &HFF00,0
HN 210 DISKREAD = &HFC00
DB 220 ' Set up
ND 230 PRINT: PRINT "Place the disk you wish to"
```

```
NM 240 PRINT "examine into drive A."
GL 250 GOSUB 1740
OF 260 CHDIR "A:\"  'tells DOS a new disk has bee
       n inserted
ON 270 POKE &HFF01,1: POKE &HFF02,0
LE 280 CALL DISKREAD
OL 290 IF PEEK(&HFF03)=0 THEN 380
AG 300 GOSUB 1500
NF 310 PRINT "Non-IBM disk or unreadable FAT."
IG 320 PRINT
GP 330 PRINT "Assumed to be 2-sided disk,"
KJ 340 PRINT "with 8 or 9 sectors per track,"
OA 350 PRINT "all sectors unallocated."
KA 360 SECTORS=9: SIDES=2: NONDATA=12
KP 370 GOSUB 1740: GOTO 520
AE 380 GOSUB 1500: I=PEEK(&HFD00)
DM 390 IF I=&HFF THEN SIDES=2: SECTORS=8: NONDATA
       =11
LD 400 IF I=&HFE THEN SIDES=1: SECTORS=8: NONDATA
       =7
GJ 410 IF I=&HFD THEN SIDES=2: SECTORS=9: NONDATA
       =12
CC 420 IF I=&HFC THEN SIDES=1: SECTORS=9: NONDATA
       =9
FJ 430 IF SECTORS=9 THEN FATSIZE=531 ELSE FATSIZE
       =476
MD 440 IF SIDES=0 THEN 310
HL 450 PRINT "Analyzing file allocation table..."
JD 460 FATP=1: GOSUB 1570
OJ 470 ' If 9-sector disk, read the rest of the F
       AT
JM 480 IF SECTORS<>9 THEN 520
JD 490 POKE &HFF01,2: CALL DISKREAD
GD 500 GOSUB 1570
JB 510 ' Initialize
DN 520 SCTR=NONDATA: STP!=&H3C00
FK 530 GOSUB 1500: COLOR 15,0
EH 540 PRINT "This is a";SIDES;"-sided, ";SECTORS
       ;"-sector disk."
GD 550 COLOR 7,0
LF 560 PRINT: PRINT "Scan begins with the first d
       ata sector."
OA 570 PRINT "Then press:": PRINT
CP 580 PRINT CHR$(27);"  to view the preceding se
       ctor"
CN 590 PRINT CHR$(26);"  to view the following se
       ctor"
PL 600 PRINT "A  to ADD the current sector to the
       "
AF 610 PRINT "       file being reconstructed"
```

```
IO 620 PRINT "N  to view the NEXT unallocated sec
       tor"
FF 630 PRINT "J  to JUMP to a specific sector"
OG 640 PRINT "Q  to QUIT."
GP 650 GOSUB 1740
JF 660 ' Display a sector
FD 670 GOSUB 1500: COLOR 15,0
EO 680 IF SCTR<0 THEN SCTR=0
JO 690 IF SCTR>32767 THEN SCTR=32767
OC 700 PRINT "Track";INT(SCTR/(9*SIDES));
BE 710 IF SIDES=2 THEN PRINT " Side";INT((SCTR MO
       D 18)/9);
HC 720 PRINT " Sector";1+(SCTR MOD 9);
BC 730 PRINT " (LS #";SCTR;")"
GN 740 COLOR 7,0: PRINT
GA 750 POKE &HFF01,SCTR MOD 256
HH 760 POKE &HFF02,INT(SCTR/256)
LH 770 CALL DISKREAD
KP 780 IF PEEK(&HFF03)=0 THEN 810
MO 790 PRINT "<< Nonexistent or unreadable sector
        >>"
JL 800 GOTO 860
JC 810 FOR I=&HFD00 TO &HFEFF
GO 820  J=PEEK(I)
JG 830  IF J>31 THEN PRINT CHR$(J);: GOTO 850
LE 840  COLOR 0,7: PRINT CHR$(J+64);: COLOR 7,0
OI 850 NEXT I
BB 860 COLOR 15,0: PRINT: PRINT
LG 870 IF INKEY$<>"" THEN 870
CH 880 PRINT CHR$(26)+" "+CHR$(27)+" Add Next Jum
       p Quit ?";
HH 890 Q$=INKEY$: IF Q$="" THEN 890
PD 900 IF Q$=(CHR$(0)+CHR$(75)) THEN SCTR=SCTR-1:
        GOTO 670
NB 910 IF Q$=(CHR$(0)+CHR$(77)) THEN SCTR=SCTR+1:
        GOTO 670
GC 920 IF Q$="A" OR Q$="a" THEN 980
OF 930 IF Q$="N" OR Q$="n" THEN 1130
GO 940 IF Q$="J" OR Q$="j" THEN 1220
BO 950 IF Q$="Q" OR Q$="q" THEN 1280
DK 960 BEEP: GOTO 890
EF 970 ' Add sector to workspace
KN 980 PRINT: PRINT "Adding..."
KM 990 STPF!=STP!
EN 1000 IF STPF!>=64512! THEN PRINT "Out of works
        pace.":GOTO 1100
DN 1010 FOR I=0 TO 511
IN 1020   POKE STP!+I,PEEK(&HFD00+I)
PF 1030 NEXT
```

```
NP 1040 STP!=STP!+512
FG 1050 PRINT "Done.  Workspace now";
CH 1060 STPF!=STP!
JE 1070 IF STPF!<0 THEN STPF!=STPF!+65536!
GC 1080 PCT!=100*((STPF!-15360)/49152!)
DK 1090 PRINT INT(PCT!+.5);"% full."
IG 1100 PRINT CHR$(26)+" "+CHR$(27)+" Next Jump Q
        uit ?";
IE 1110 GOTO 890
DD 1120 ' Go to next free sector
FF 1130 GOSUB 1500
GG 1140 PRINT "Searching for unallocated sector..
        ."
CP 1150 SCTR=SCTR+1: LOCATE 8,1: PRINT SCTR: GOSU
        B 1660
KF 1160 IF (FATV=0) AND (SCTR<=719) THEN 670
LI 1170 IF SCTR<719 THEN 1150
AC 1180 PRINT: PRINT "No more unallocated sectors
        ."
GK 1190 PRINT: COLOR 15,0
OK 1200 PRINT CHR$(26)+" "+CHR$(27)+" Jump Quit ?
        ": GOTO 890
HC 1210 ' Jump to a particular sector
LP 1220 PRINT: INPUT "Track ( 0 to 39 )";T
JK 1230 H=0: IF SIDES>1 THEN INPUT "Side ( 0 or 1
         )";H
BL 1240 PRINT "Sector ( 1 to";SECTORS;") ";
HF 1250 INPUT S
JA 1260 SCTR = T*SIDES*9 + H*9 + S - 1
HK 1270 GOTO 670
FE 1280 ' Write out workspace to file
JI 1290 IF STP!=&H3C00 THEN 1470
CI 1300 GOSUB 1500: PRINT
BF 1310 PRINT "Ready to create a disk file from"
MF 1320 PRINT "contents of workspace."
QM 1330 PRINT
GP 1340 PRINT "Place the disk you wish to use int
        o"
GK 1350 PRINT "the appropriate drive (drive A is
        OK)."
PH 1360 GOSUB 1730: GOSUB 1500
KF 1370 PRINT "Enter name of file to be created:"
GP 1380 INPUT N$:IF INSTR(N$,".") THEN 1390
DF 1385 N$=N$+".BAS"
JA 1390 OPEN N$ FOR OUTPUT AS #1
MI 1400 PRINT: PRINT "Writing..."
HP 1410 FOR I!=&H3C00 TO STP!
GE 1420   PRINT #1,CHR$(PEEK(I!));
PM 1430 NEXT I!
PI 1440 CLOSE #1: PRINT "All done."
```

```
PM 1450 FOR DELAY=1 TO 1000: NEXT
BD 1460 ' End program
MK 1470 COLOR 7,0: POKE 92,24: CLS: KEY ON
IE 1480 CLEAR,&HFF10: END
BO 1490 ' Subroutine -- display header
PD 1500 CLS: COLOR 7,0
DP 1510 A$=CHR$(186): B$=STRING$(38,205)
JP 1520 PRINT CHR$(201);B$;CHR$(187);
OB 1530 PRINT A$;"                IBM Disk Rx
           ";A$;
NM 1540 PRINT CHR$(200);B$;CHR$(188)
JE 1550 RETURN
MH 1560 ' Subroutine -- read FAT
NK 1570 FOR BYTE=&HFD00 TO &HFEFF
KB 1580  FAT(FATP)=PEEK(BYTE)
OJ 1590  FATP=FATP+1
IG 1600  IF FATP>FATSIZE THEN RETURN
EH 1610 NEXT BYTE
JN 1620 RETURN
BI 1630 ' Subroutine
IL 1640 '  accepts SCTR (sector number)
DL 1650 '  returns FATV (value of its FAT entry)
MN 1660 C=2+INT((SCTR-NONDATA)/SIDES)
LL 1670 IF (C<1) OR (SCTR>719) THEN FATV=&HFF0:RE
        TURN
GE 1680 I=INT(C*1.5)+1
EF 1690 FATV = FAT(I) + 256*(FAT(I+1) AND &HF)
DA 1700 IF (C MOD 2)=0 THEN RETURN
JN 1710 FATV = INT(FAT(I)/16) + 16*FAT(I+1)
JP 1720 RETURN
GK 1730 ' Subroutine -- wait for keystroke
EL 1740 WHILE INKEY$<>"": WEND
BK 1750 PRINT
CF 1760 PRINT "(Press any key to continue)";
PA 1770 WHILE INKEY$="": WEND
KB 1780 RETURN
```

CHAPTER 2

Recreation

Martian Prisoner

Alan Poole
Translation by Gregg Peele

"Martian Prisoner" is a mini-adventure game for the IBM PC and PCjr. If you've never played an adventure game before, this is a good introduction. Unlike most computer games, text adventures have no graphics and do not require fast reflexes—instead, they test the player's patience and cunning.

Without warning, the Martians have suddenly started an all-out war against Earth. They have captured you and are holding you prisoner in a cell on a Martian space cruiser headed toward Earth. The cruiser also carries a secret weapon that can neutralize all of Earth's defenses. Your task is to destroy the Martian ship and escape in a lifecraft before the Martians can complete their sinister mission.

Like Radio Dramas

Adventure games require you to solve puzzles and explore a simulated world inside the computer. The computer will describe what you see and what happens, and you tell the computer what you want to do. Instead of using screen graphics, adventure games rely on text descriptions and your imagination. It's like the difference between old-time radio dramas and television; despite the visual impact of video, the mind can still imagine a scene more exciting than a camera can picture.

In "Martian Prisoner," you start off in the prison cell of the Martian space cruiser. Besides the cell, the cruiser contains several other rooms. It's up to you to explore the rooms and find a way to destroy the ship. In each room, the computer will describe your surroundings and list the objects in the room. The computer then waits for you to type a command, consisting of one to three words.

For example, you could type GO NORTH to move north. If there is an object—say, a Geiger counter—in the room, you could type GET GEIGER COUNTER to pick it up. Type INVENTORY at any time to see a list of the objects you are

carrying. All commands and nouns can be abbreviated to the first three letters. You can list your inventory by typing INV, for instance.

Although Martian Prisoner is a short adventure game, you must solve several puzzles to win. If you've never played an adventure game, working with Martian Prisoner is a good way to prepare for the more elaborate adventure games available.

Commands for Martian Prisoner

GO NORTH	DROP	INVENTORY
GO SOUTH	EAT	KILL
GO EAST	GET	OPEN
GO WEST	HIT	READ
		WEAR

Martian Prisoner

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
NJ 100 WIDTH 40:KEY OFF
PN 110  CLS:GOSUB 960
GB 120 GOSUB 170:IF R=6 AND U=0 THEN R=1:GOTO 120
GL 130 GOSUB 410
FK 140 PRINT :ON V GOSUB 550,550,550,550,560,690,
       730,740,770,800,820,840,860
KI 150 IF V=14 THEN GOSUB 860
CO 160 GOTO 120
CI 170 PRINT :ON R GOSUB 220,260,270,280,320,330,
       370,380
IF 180 IF I(4)=-1 AND I(6)=-1 THEN PRINT NS$
GM 190 PRINT :PRINT "Objects:":PRINT
LJ 200 FOR L=1 TO 8:IF I(L)=R THEN PRINT TAB(8);N
       $(L)
LF 210 NEXT :PRINT :RETURN
IC 220 PRINT "You are in a prison cell."
HG 230 IF RND(1)>.25 THEN RETURN
FJ 240 G=1:PRINT "A guard has turned off the forc
       e field  and entered the cell.":C%(1,1)=2
MG 250 RETURN
CO 260 PRINT "You are in a N/S hall.":RETURN
NN 270 PRINT "You are in the engine room.":RETURN
ND 280 PRINT "You are in a small room. A large si
       gn   is on the wall."
FA 290 IF I(4)=-1 THEN PRINT NS$
MK 300 IF I(6)=-1 THEN 880
MP 310 RETURN
CL 320 PRINT "You are in the supply room.":RETURN
DK 330 PRINT "You are in the north side of the ha
       ll."
```

```
CC 34Ø IF U=1 THEN PRINT "The guards don't notice
        you."
CM 35Ø IF U=Ø THEN PRINT "The guards take you bac
       k to the cell.":G=Ø
NJ 36Ø RETURN
PO 37Ø PRINT "You are in a large room.":RETURN
KP 38Ø PRINT "You are in a strange garden where f
       ood  is grown for the crew."
FB 39Ø IF I(4)=-1 THEN PRINT NS$
MO 4ØØ RETURN
JJ 41Ø C$="":N=Ø:V=Ø:PRINT :INPUT "Command";C$:PR
       INT "":IF C$=""THEN 41Ø
HM 42Ø P=Ø:IF LEN(C$)<2 THEN 46Ø
HM 43Ø FOR L=2 TO LEN(C$)-1
DA 44Ø IF MID$(C$,L,1)=" "THEN P=L
OK 45Ø NEXT
NI 46Ø IF P=Ø THEN V$=C$:N$=""
QJ 47Ø IF P>Ø AND P=LEN(C$)THEN V$=C$:N$=""
IG 48Ø IF P>Ø AND P<LEN(C$)THEN V$=LEFT$(C$,P-1):
       N$=RIGHT$(C$,LEN(C$)-P)
CJ 49Ø FOR L=1 TO 14:IF LEFT$(V$,3)=V$(L) THEN V=
       L
HK 5ØØ NEXT:FOR L=1 TO 8:IF LEFT$(N$,3)=A$(L)THEN
        N=L
GL 51Ø NEXT:IF N>Ø AND V>Ø THEN RETURN
HG 52Ø IF N=Ø AND V>Ø AND N$=""THEN RETURN
JE 53Ø IF N=Ø AND V=5 THEN RETURN
JM 54Ø PRINT :PRINT "I don't understand.":GOTO 41
       Ø
IO 55Ø N$=V$:GOTO 57Ø
OK 56Ø N$=LEFT$(N$,1)
KO 57Ø IF R=1 AND N$="E" AND G=Ø THEN PRINT "The
       force field stops you.":RETURN
BJ 58Ø IF R<>1 OR N$<>"E"OR G=Ø THEN 6ØØ
JH 59Ø PRINT "As you leave the cell the force fie
       ld isactivated, trapping the guard."
ME 6ØØ IF R=2 AND N$="E" AND C%(2,1)=Ø AND I(8)>-
       1 THEN PRINT "The locked door stops you.":
       RETURN
BC 61Ø IF R<>2 OR N$<>"E"OR C%(2,1)>Ø THEN 63Ø
BP 62Ø PRINT "You unlock the door with the key.":
       C%(2,1)=5:N$(7)="open door"
OJ 63Ø IF N$="N"THEN D=Ø
IE 64Ø IF N$="E"THEN D=1
ED 65Ø IF N$="S"THEN D=2
JK 66Ø IF N$="W"THEN D=3
KJ 67Ø IF C%(R,D)=Ø THEN PRINT CN$:RETURN
BM 68Ø PRINT "Ok":R=C%(R,D):RETURN
GH 69Ø IF N=1 OR N=2 OR N=3 OR N=7 THEN PRINT "Yo
       u can't lift it!":RETURN
```

```
BH 700 IF I(N)<>R THEN PRINT "It's not here.":RET
       URN
HP 710 IF N=5 THEN 830
QP 720 PRINT "Ok":I(N)=-1:RETURN
MK 730 PRINT "Ok":I(N)=R:RETURN
AE 740 PRINT "You are carrying:"
BL 750 FOR L=1 TO 8:IF I(L)=-1 THEN PRINT TAB(3);
       N$(L)
KN 760 NEXT :RETURN
OH 770 IF N<>3 OR R<>4 THEN PRINT CN$:RETURN
FM 780 PRINT "Atomic fuel near this room. Do not
       bringany other radioactive materials into"
EE 790 PRINT "this room.":RETURN
EC 800 IF N<>7 OR R<>2 OR I(8)>-1 THEN PRINT CN$:
       RETURN
EM 810 N$="E":GOTO 620
OC 820 IF I(5)<>R THEN PRINT CN$:RETURN
DJ 830 PRINT "You are now wearing a uniform.":I(5
       )=-1:U=1:RETURN
GD 840 IF N<>6 THEN PRINT RI$:RETURN
GH 850 PRINT "You quickly become sick and die.":G
       OTO 910
ME 860 IF N=2 THEN PRINT "The guard shoots you.":
       GOTO 910
EI 870 PRINT CN$:RETURN
GM 880 PRINT :PRINT "The radioactive plant suppli
       es enough ":PRINT "neutrons to start a cha
       in reaction."
JJ 890 PRINT "The whole ship blows up. You escape
        in alifecraft."
NB 900 PRINT :PRINT "YOU WIN !!":GOTO 920
DF 910 PRINT :PRINT "YOU LOSE !"
NM 920 PRINT :PRINT :PRINT "Another game?"
FF 930 K$=INKEY$:IF K$="Y" THEN RUN
NB 940 IF K$="N"THEN END
IP 950 GOTO 930
QM 960 LOCATE 10,13:PRINT "MARTIAN PRISONER"
PE 970 LOCATE 12,9 :PRINT "(Caps Lock must be dow
       n)"
BN 980 LOCATE 20,10:PRINT "Press any key to start
       "
GG 990 A$=INKEY$:IF A$="" THEN A=RND(1):GOTO 990
       ELSE CLS
BN 1000 DIM V$(14),C%(8,3),I(8),N$(8),A$(8)
LF 1010 R=1:FOR L=1 TO 14:READ V$(L):NEXT
NI 1020 FOR L=1 TO 8:READ C%(L,0),C%(L,1),C%(L,2)
        ,C%(L,3):NEXT
GN 1030 FOR L=1 TO 8:READ N$(L),A$(L),I(L):NEXT
DN 1040 CN$="You can't.":RI$="Don't be ridiculous
        !"
```

```
GM 1Ø5Ø NS$="The geiger counter is clicking.":RET
        URN
ON 1Ø6Ø DATA N,E,S,W,GO,GET,DRO,INV,REA,OPE,WEA,E
        AT,KIL,HIT
GI 1Ø7Ø DATA Ø,Ø,Ø,Ø,6,Ø,3,Ø,2,4,Ø,Ø,Ø,Ø,Ø,3,Ø,Ø,
        Ø,2,7,Ø,2,Ø,Ø,8,6,Ø,Ø,Ø,Ø,7
EE 1Ø8Ø DATA FORCE FIELD,FOR,1,GUARDS,GUA,6,SIGN,
        SIG,4
JC 1Ø9Ø DATA GEIGER COUNTER,COU,5,UNIFORM,UNI,5
LL 11ØØ DATA PLANT,PLA,8,LOCKED DOOR,DOO,2,MAGNET
        IC KEY,KEY,3
```

Aardvark Attack

Todd Heimarck
Translation by Tim Victor

So, you think you're pretty quick on the keyboard? Here's a game that will test anybody's skill. "Aardvark Attack" has ten levels of difficulty, making it a challenge for beginners and experts alike. It runs on any PC or PCjr.

Earth is being attacked again.

Mutant aardvarks from Andromeda are dropping alphanumeric bombs. There are 26 types of bombs, each requiring a different defense. Typing the letter Q, for example, will set up the Q-defense against Q-bombs. The same applies to the other 25 letters. Also, the aardvarks are attacking your ten biggest cities (numbered 1 through 0). Once you set up the defense, you have to decide which city is being attacked.

A Typing Tester

You could say that "Aardvark Attack" helps you find your way around the keyboard. It won't teach you how to type, but it's good practice if you are just learning. From that viewpoint it's educational.

If you already know how to type, you will find it a challenging game. Aardvark Attack uses the entire keyboard instead of a joystick. Fanatic game players should enjoy it.

How to Play

Your goal is first to type the correct letter and then the correct number. The letter appears in a radar "window" in the upper-right corner of the screen. When it appears, you respond by typing it. If you make a mistake, try again. When you get the right letter, the falling bomb changes shape. Then look at the row of numbers at the bottom of the screen—representing your cities—and type the corresponding number to destroy the bomb.

You have limited time before the oncoming bomb hits the ground, ranging from a few seconds at the novice level to only a fraction of a second at the expert levels. The faster games score more points because they're more difficult.

Destroying a bomb before it reaches the middle of the screen scores double the usual number of points. If you repel a bomb in the top quarter of the screen, you get a triple-point bonus. Furthermore, points are doubled for each higher level.

The game can be paused while a bomb is falling by pressing the] key (right bracket). To continue, press Enter. To change levels during a game, press the number key of the desired level after pausing with the] key. Pressing the E key while the game is paused ends the program.

If you are a parent or teacher of young children who are just learning their letters, you could act as their fingers. Have them call out the name of the letter for you to type. Beginning typists can practice at the lower levels, while expert typists and game players will prefer the higher levels. It's a game for almost everyone.

Aardvark Attack

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
DF 5 SCREEN 0,1
DK 10 CB=7:CI=12:CP=11:CS=2:CL=15
KD 20 WIDTH 40:COLOR 7,0,0:CLS:LOCATE 1,2,0:KEY O
      FF:DEF SEG=0: POKE 1047,96
ED 30 DIM P$(1):P$(0)=CHR$(15):P$(1)="o":SC=0:HI=
      0:D=1:DIM D$(6):
FI 40 COLOR CS:LOCATE 3,12:PRINT "AARDVARK ATTACK
      "
JI 50 NL=41:NS=32767:SOUND NL,10:SOUND NS,1:SOUND
       NL,4:SOUND NL*1.19,10:SOUND NL,10
IK 60 LOCATE 22,10:PRINT "Press any key to begin"
NA 70 A$=INKEY$:A=RND:IF A$="" THEN 70
KF 80 '          DRAW SCREEN
OE 90 S$=STRING$(5,177)+CHR$(216)+STRING$(12,32)+
      CHR$(216)+STRING$(6,177)+STRING$(13,176)
HB 100 COLOR CB
BP 110 FOR I=1 TO 24:LOCATE I,1:PRINT S$;:NEXT
CN 120 COLOR CP
KL 130 LOCATE 20,8:PRINT "1234567890";:FOR I=21 T
       O 24:LOCATE I,8:PRINT STRING$(10,176):NEXT
EF 140 COLOR CS
OE 150 LOCATE 19,27:PRINT "Level        ";
PF 160 LOCATE 20,27:PRINT "             ";
PL 170 LOCATE 21,27:PRINT "Score        ";
BD 180 LOCATE 22,27:PRINT "             ";
NB 190 LOCATE 23,27:PRINT "High         ";
KN 200 GOSUB 560
NO 210 FOR I=3 TO 11:LOCATE I,28:PRINT "
       ";:NEXT
```

```
EC 220 COLOR CS
JF 230 GOSUB 730
EL 240 LOCATE 1,1:PRINT "Enter difficulty level (
       0=Easy/9=Hard)"
LC 250 GOSUB 660
MK 260 '        CHOOSE BOMB AND CITY
IA 270 COLOR CB
OL 280 LOCATE 1,1:PRINT S$;
LH 290 Z=(RND*26000):Z=INT(Z-INT(Z/26)*26):RESTOR
       E:FOR J=1 TO Z:READ A1,A2,A3,A4,A5,A6,A7:N
       EXT
GE 300 FOR I=0 TO 6:D$(I)="       ":READ A%:FOR J
       =1 TO 7:B%=(A%>63):MID$(D$(I),J)=CHR$(32-B
       %*187):A%=(A%+B%*64)*2:NEXT:NEXT
NG 305 COLOR CI
QI 310 FOR I=1 TO 40:N=INT(RND*10):IF N THEN LOCA
       TE 1,N+7 ELSE LOCATE 1,17
EB 330 PRINT CHR$(178);:SOUND 261.6*2^(N/12),1:NE
       XT
HI 340 LY=0
PF 350 COLOR CL
CK 360 FOR I=4 TO 10: LOCATE I,29:PRINT D$(I-4);:
       NEXT
NF 370 COLOR CI
CN 380 '       THE AARDVARKS ATTACK
CA 390 IF N THEN C=N+7 ELSE C=17
KG 400 R=2:WHILE SCREEN(R,C)=32 :LOCATE R,C:PRINT
        P$(LY);:SOUND 50*(24-R),1:FOR K=1 TO HF:A
       $=INKEY$
HB 410 IF A$="" THEN 450
JH 420 IF A$="]" THEN COLOR CS:GOSUB 660:COLOR CI
       :GOTO 450
QN 430 IF LY=1 THEN IF ASC(A$)-48=N THEN COLOR CS
       :GOSUB 530:COLOR CI:GOTO 490
ML 440 IF LY=0 THEN IF ASC(A$)-65=Z THEN LY=1
GP 450 NEXT:R=R+1:WEND
EE 460 '       ATTACK SUCCESSFUL
QO 470 LOCATE R,C:PRINT P$(LY);:IF R=24 THEN 630
CK 480 IF LY=0 THEN R=R+1:LOCATE R,C:PRINT P$(LY)
       ;:IF R=24 THEN 630
CF 485 FOR I=1 TO 2:SOUND 50*(24-R),2:SOUND NS,2:
       NEXT:SOUND 50*(24-R),5
DA 490 FOR I=2 TO R:LOCATE I,C:PRINT " ";:NEXT
NP 500 FOR I=3 TO 11:LOCATE I,28:PRINT "
       ";:NEXT
FO 510 GOTO 260
PF 520 '     ATTACK THWARTED
OC 530 SC=SC+2^(D-1)
BH 535 SOUND 164.8,4:SOUND NS,1:SOUND 164.8,2:SOU
       ND 220,10
```

```
AB 540 IF R<6 THEN SC=SC+2^(D-1)
IN 550 IF R<12 THEN SC=SC+2^(D-1)
HI 560 B$=STR$(SC):B$=LEFT$("00000",6-LEN(B$))+MI
       D$(B$,2,LEN(B$)-1)
NK 570 LOCATE 21,33:PRINT B$;
PL 580 IF SC>HI THEN HI=SC
DD 590 B$=STR$(HI):B$=LEFT$("00000",6-LEN(B$))+MI
       D$(B$,2,LEN(B$)-1)
OJ 600 LOCATE 23,33:PRINT B$
MC 610 RETURN
PF 620 '     GAME OVER
BI 630 SC=0
PA 635 FOR J=1 TO 12:FOR I=400 TO 240 STEP -80:SO
       UND I,1:NEXT:LOCATE J*2,1:PRINT "
                                            ";:NEXT
BC 637 LOCATE 1,1:PRINT "               Play again? (Y
        OR N)          "
IG 638 A$=INKEY$:IF A$="" THEN 638
QM 639 IF A$="N" THEN A$="E":GOTO 680
BJ 640 GOTO 100
DP 650 '     SET DIFFICULTY
CH 660 A$=INKEY$:IF A$<>"" THEN 660
PG 670 A$=INKEY$:IF A$="" THEN 670
JM 680 IF A$="E" THEN POKE 1047,64:CLS:COLOR 7:EN
       D
KE 690 IF ASC(A$)=13 THEN 750
CD 700 IF ASC(A$)<48 THEN 670
EC 710 IF ASC(A$)>57 THEN 670
EJ 720 D=ASC(A$)-47
FE 730 HF=INT(100/(D*D))-4*D+40
FE 740 LOCATE 19,35:PRINT MID$(STR$(D-1),2,1);
CF 750 A$=INKEY$:IF A$<>"" THEN 750
NN 760 RETURN
JO 1001 DATA 24,60,102,102,126,102,102
AG 1002 DATA 126,51,51,63,51,51,126
BJ 1003 DATA 30,51,96,96,96,51,30
PH 1004 DATA 124,54,51,51,51,54,124
GP 1005 DATA 127,49,52,60,52,49,127
GH 1006 DATA 127,49,52,60,52,48,120
FM 1007 DATA 30,51,96,96,103,51,15
QM 1008 DATA 102,102,102,126,102,102,102
IN 1009 DATA 60,24,24,24,24,24,60
CP 1010 DATA 15,6,6,6,102,102,60
PP 1011 DATA 115,51,54,60,54,51,115
KD 1012 DATA 120,48,48,48,49,51,127
MO 1013 DATA 99,119,127,127,107,99,99
CL 1014 DATA 99,115,123,111,103,99,99
HH 1015 DATA 28,54,99,99,99,54,28
ID 1016 DATA 126,51,51,62,48,48,120
MG 1017 DATA 60,102,102,102,110,60,14
```

```
BH 1018 DATA 126,51,51,62,54,51,115
AH 1019 DATA 60,102,48,24,12,102,60
JD 1020 DATA 126,90,24,24,24,24,60
DO 1021 DATA 102,102,102,102,102,102,126
GG 1022 DATA 102,102,102,102,102,60,24
EE 1023 DATA 99,99,99,107,127,119,99
ED 1024 DATA 99,99,54,28,28,54,99
BK 1025 DATA 102,102,102,60,24,24,60
LD 1026 DATA 127,99,70,12,25,51,127
```

Laser Barrage

Sean Igo
IBM Version by Kevin Mykytyn

Here's a frantic action game that's sure to leave you sweating. "Laser Barrage" runs on any IBM PC or PCjr. On a PC without the color/graphics adapter, the 40-column display occupies only half of the monochrome screen.

The thermometer says it's 117 degrees outside, and you can believe it. Even though your tractor is air-conditioned, it's sweltering in the cab. And to think that Grandpa used to dig potatoes in that heat—by hand, with a hoe. Whatever a hoe was.

But that was before the oil ran out. When the last drop was gone, the engineers had to come up with something fast. They'd been working on robots, and they'd made some progress, too. Some of those robots could almost *think.*

But when the engineers all turned to biology, the robots were forgotten. While the gene-splicers were inventing new hybrid cucumbers to gather and store solar energy in huge pods, the robots wandered off to the hills.

You stop to wipe the sweat from your brow—and that's when you spy another one. Darned robots! Why can't they get their energy somewhere else? You've worked hard all spring, planting those solar cucumbers, and now that the energy pods are ripe, those metal maniacs are coming from miles around, hiding behind the hedgerows and grabbing energy pods when they can.

With a sigh, you activate the tractor's roof-mounted laser shotgun. It's time to zap some robots. Worse than rabbits in a bean patch, like Grandpa used to say.

Whatever a rabbit was....

Playing Laser Barrage

"Laser Barrage" is a fast-paced game that requires both skill and strategy. The object is to defend your solar energy pods by using your laser to clear out those bothersome robots—without getting zapped yourself. You can get zapped by

accidentally running into your own energy pods (which are considered delicacies by the robots), colliding with a robot, or crashing your tractor into one of many hedgerow barriers. It's an exciting game of pursuit, and the prizes are your precious energy pods.

But Laser Barrage is more than high-tech tag. Your robot opponents actually have some intelligence, and they'll use it to find a way around obstacles or to home in on the pods you're so valiantly trying to protect. As you'll quickly discover, it takes more than a quick trigger finger to run up high scores. If you're going to outfox those robots, strategy is important, too.

Game play is straightforward. Use the cursor keys to move your gleaming yellow laser-equipped tractor north, south, east, or west; press the numeral 5 key to fire bright orange laser bolts. You can also move diagonally by pressing 1, 3, 7, or 9, although the PC's keypad makes this easier than on the PCjr. You can get continuous motion (or automatic sequential fire) by holding down the appropriate key, since all IBM keys automatically repeat. But if you overflow the keyboard buffer, the computer will beep in protest and the keyboard won't respond until the buffer clears.

123
456
789

You start with three laser-equipped tractors and one robot opponent. The screen displays the number of tractors you have left as well as your accumulated score. The border color changes each time you advance a level. There are 22 different levels, each with three more robot opponents than the one before it. Surviving the higher levels is a real challenge in Laser Barrage.

Scoring

Laser Barrage rewards proficiency and shooting skills. You'll find that as your skills improve, so does your score.

The score depends on three factors: the number of robots hit, how many shots it took to get them, and how many pods are left intact at the end of each round. You lose one point for every shot, but you gain five points each time a shot connects and a robot is destroyed.

When all the robots have been removed, you are also awarded points for each remaining pod. The bonus is 10 points per pod on level one, 40 points per pod on level two, 70 points per pod on level three, and so on. In other words, on each successive level you'll earn 30 additional points per remaining pod.

Since the robots have some degree of intelligence, they are somewhat predictable—and you can take advantage of that characteristic to improve your scores. Once a robot has locked onto a pod, for instance, it will continue moving toward that pod as long as nothing intervenes. Line yourself up between the pod and the approaching robot, and you'll have an excellent chance of success. During program testing, some players have scored more than 5000 points.

Customizing the Game

Since IBM computers do not have sprites, graphics characters in the standard character set have been used for robots, pods, and tractors. That makes it easy to customize the game, to some extent, by using characters different from the ones originally chosen.

For example, Laser Barrage uses ASCII character 15 for the pods and character 232 for the robots. Those particular characters are defined in line 270 by CHR$(15) and CHR$(232), and in line 290 by variables POD and RO. By changing the numbers you can change the characters. Replace CHR$(15) with CHR$(127), and POD=15 with POD=127, and you'll be protecting houses instead of power pods.

Similarly, replace CHR$(232) with CHR$(21) (and RO=232 with RO=21) to turn the robots into sinister snake-like creatures. You can use any characters you choose, but be sure to change CHR$ in lines 270 *and* 290. You can also change the maximum number of robots, increasing or decreasing it from the preset value of 64. To modify the robot population, change the 64 in DIM RP(64,3) in line 290. You can substitute any number you desire, and the maximum will change accordingly.

Inside Laser Barrage

The game incorporates several interesting programming techniques which PC and PCjr owners may find useful. Speed is improved by locating the main program loop at the beginning of the listing; subroutines are placed at the end, where they will not slow things down.

Speed is increased even further by performing all numeric operations with integer variables instead of real variables. Both the PC and PCjr handle integer variables much more efficiently than they can handle real variables, and the result is a dramatic increase in smoothness and playability.

To see how integer variables increase execution speed, try the following short counting experiment to see how long it takes your computer to count to 50,000. Type in FOR A=1 TO 50000:NEXT and press Enter. Note how long it takes using a real variable. Then try it with an integer variable, using the line FOR A%=1 TO 50000:NEXT. Integer variables allow your computer to count almost twice as fast! You could increase speed even further by replacing the numbers 1 and 50000 with integer variables B% and C% and defining those variables before executing the speed check.

One of the most interesting aspects of Laser Barrage is its intelligent robots. Lines 70–110 give the robots their mental abilities. In effect, each robot evaluates the position of its chosen target, looks at its own location, and then determines the best path to follow to reach the target. Pod positions are stored in array PP, while robot positions are held in array RP. The X and Y components of those positions are compared, and the SGN operation is then used to yield a value of −1 (if the robot needs to move up or left) or 1 (if the robot needs to go down or right). If the optimum move would place the robot on an occupied space, then a random move is chosen. In either case, once a robot reaches and consumes a pod, it immediately goes after another one.

Just like rabbits in a bean patch.

Laser Barrage

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
CH 1Ø COLOR Ø,Ø,Ø:GOSUB 27Ø:GOSUB 57Ø:CLS:GOSUB 3
      5Ø:GOTO 5Ø
PJ 2Ø A$=INKEY$:JA=VAL(A$):IF JA=N5 THEN  GOSUB 1
      4Ø ELSE IF JA THEN LOCATE JY,JX:PRINT BL$:J
      Y=JY+J(JA,N1):JX=JX+J(JA,N2):IF SCREEN(JY,J
      X)<>BL THEN 47Ø ELSE LOCATE JY,JX:COLOR N14
      :PRINT MA$(JA):POKE MM,PEEK(NN):JB=JA
CK 3Ø CO=CO+N1:IF CO<N3  THEN 2Ø
KL 4Ø CO=NØ
FE 5Ø A=A+N1:IF A>NR THEN A=N1
NE 6Ø C=RP(A,N3):RX=RP(A,N2):IF RX=NØ THEN 5Ø
HB 7Ø RY=RP(A,N1):PX=PP(C,N2):PY=PP(C,N1)
CG 8Ø DX=SGN(PX-RX):DY=SGN(PY-RY)
QN 9Ø P=SCREEN(RY+DY,RX+DX)
FB 1ØØ IF P=POD THEN GOSUB 2ØØ:GOTO 12Ø
PJ 11Ø IF P<>BL  THEN DX=RND(N1)*N2-N1:DY=RND(N1)
       *N2-N1:GOTO 9Ø
CF 12Ø RP(A,N1)=RY+DY:RP(A,N2)=RX+DX
```

```
AG 13Ø LOCATE RY,RX:PRINT BL$:LOCATE RP(A,N1),RP(
       A,N2):COLOR N4:PRINT RO$:IF LOSE THEN 47Ø
       ELSE 2Ø
NH 14Ø NS=NS+N1:COLOR 12:OFX=N2:OFY=N2:FOR B=N1 T
       O N1Ø:SOUND B*N4Ø,.6:FY=JY+B*J(JB,N1):FX=J
       X+B*J(JB,N2):SC=SCREEN(FY,FX)
BO 15Ø IF SC<>RO THEN 17Ø ELSE FOR W= N1 TO NR:IF
        RP(W,N1)=FY AND RP(W,N2)=FX THEN RP(W,N2)
       =NØ:NK=NK+N1:SCORE=SCORE+N5:LOCATE 25,8:PR
       INT SCORE;:IF NK=NR THEN WIN=NM1:W=NR
OJ 16Ø NEXT
DH 17Ø IF SC=POD OR SC=WALL THEN B=N1Ø:ELSE LOCAT
       E FY,FX:PRINT MA$(JB):LOCATE OFY,OFX:PRINT
        BL$;:OFX=FX:OFY=FY
PF 18Ø IF SC=RO THEN B=N1Ø
PP 19Ø NEXT:LOCATE OFY,OFX:PRINT BL$:IF WIN THEN
       52Ø ELSE RETURN
HO 2ØØ SOUND N4Ø,N2:TY=RY+DY:TX=RX+DX:FOR E=N1 TO
        N5:IF  PP(E,N1)=TY AND PP(E,N2)=TX THEN D
       =E
NA 21Ø NEXT
BN 22Ø PP(D,N1)=NØ:PN=PN-N1:IF PN=NØ THEN LOSE=N1
       :RETURN
OE 23Ø FOR Q=N1 TO NR:IF RP(Q,N3)=D THEN GOSUB 25
       Ø
GK 24Ø NEXT:RETURN
IN 25Ø R=RND(N1)*N4+N1:IF PP(R,N1)=NØ THEN 25Ø
DC 26Ø RP(Q,N3)=R:RETURN
QO 27Ø WALL$=CHR$(178):POD$=CHR$(15):BL$=CHR$(32)
       :RO$=CHR$(232)
FF 28Ø DEFINT A-Z:WIDTH 4Ø:SCREEN Ø,1:DEF SEG=Ø:K
       EY OFF:POKE 1Ø47,96
BI 29Ø N1=1:N2=2:N3=3:NØ=Ø:N6=6:N2ØØØ=2ØØØ:N3ØØØ=
       3ØØØ:N1ØØ=1ØØ:N4Ø=4Ø:N24=24:WALL=178:BL=32
       :POD=15:DIM PP(5,2):DIM RP(64,3):NR=1:NP=5
       :N5=5:PN=NP:N14=14:N4=4:N1Ø=1Ø:NM1=-1:RO=2
       32:MM=1Ø52:NN=1Ø5Ø:N9=9:SCORE =NØ:LI=N4:LE
       =Ø
OD 3ØØ DIM J(9,2):FOR A=1 TO 9:FOR B=1 TO 2:READ
       J(A,B):NEXT:NEXT
NE 31Ø DATA 1,-1,1,Ø,1,1,Ø,-1,Ø,Ø,Ø,1,-1,-1,-1,Ø,
       -1,1
CM 32Ø FOR A=1 TO 9:READ B:MA$(A)=CHR$(B):NEXT
AF 33Ø DATA 47,25,92,27,26,26,92,24,47
MF 34Ø RETURN
MD 35Ø COLOR 15:FOR A=N1 TO 23  STEP 22:FOR B=N1
       TO N4Ø:LOCATE A,B:PRINT WALL$;:NEXT:NEXT:F
       OR B=N1 TO N4Ø STEP 39:FOR A=N1 TO 23:LOCA
       TE A,B:PRINT WALL$;:NEXT:NEXT
DJ 36Ø FOR I=N1 TO 12:GOSUB 45Ø:PRINT WALL$:NEXT
```

```
CH 370 FOR I=N1 TO N5:GOSUB 450:COLOR N10:PRINT P
       OD$:PP(I,N1)=B:PP(I,N2)=A:NEXT
JN 380 FOR I=N1 TO NR
OL 390 GOSUB 450:RP(I,N2)=A:RP(I,N1)=B:COLOR N4:P
       RINT RO$:NEXT
CM 400 JY=2:JX=2:COLOR N14:LOCATE JY,JX:PRINT MA$
       (2)
BL 410 FOR I=N1 TO NR:RP(I,N3)=RND(N1)*N4+N1:NEXT
IG 420 COLOR 14,0:LOCATE 25,1:PRINT "Score="SCORE
       ;
FD 430 COLOR 12,0:LOCATE 25,20:PRINT "Lives="LI-N
       1;
ML 440 NS=0:COLOR 15,0,LE+1:RETURN
DG 450 B=RND(1)*18+3:A=RND(1)*32+4:IF SCREEN(B,A)
       <>BL THEN 450
ON 460 LOCATE B,A,N0:RETURN
JA 470 COLOR N14:FOR C1=N1 TO N4:FOR C2=N1 TO N4:
       LOCATE JY,JX:PRINT MA$(C2):SOUND C2*100,1:
       NEXT:NEXT
MK 480 FOR TD=1 TO 2000:NEXT:POKE MM,PEEK(NN)
KB 490 LI=LI-1:IF LI>1 THEN 560 ELSE LOCATE 10,13
       :PRINT "    GAME OVER    ":LOCATE 12,10:PRIN
       T "    PLAY AGAIN? (Y/N)    "
PB 495 LOCATE 25,27:COLOR 12:PRINT"0";
GF 500 A$=INKEY$:IF A$="Y" THEN RUN
KN 510 IF A$=""THEN 500
DE 515 IF A$<>"N" THEN 500
FK 516 CLS:END
CP 520 FOR C1=N1 TO N10:COLOR 15,C1,C1+N1:FOR TD=
        1 TO 100:NEXT:SOUND C1*150,1:NEXT:LE=LE+1
       :IF LE>14 THEN LE=0
EI 530 FOR C1=N10 TO N1 STEP NM1:COLOR 15,C1,C1+N
       1:FOR TD= 1 TO 100:NEXT:SOUND C1*150,1:NEX
       T:FOR TD=1 TO 1100:NEXT
MA 540 FOR C1=1 TO 5:IF PP(C1,1)<>0 THEN LOCATE P
       P(C1,1),PP(C1,2):COLOR 9:PRINT POD$;:SOUND
        300,3:FOR TD=1 TO 500:NEXT:COLOR 10,0:PRI
       NT CHR$(29);POD$:SOUND 150,3:FOR TD=1 TO 5
       00:NEXT:SCORE=SCORE+NR*N10-NS*N1:NS=N0:LOC
       ATE 25,1:COLOR 14,0:PRINT "score="SCORE;
IK 550 NEXT:NR=NR+3
CO 560 FOR TD=1 TO 1000:NEXT:CLS:GOSUB 350:NK=0:P
       N=5:SC=BL:WIN=0:LOSE=0:POKE MM,PEEK(NN):GO
       TO 50
LD 570 CLS:COLOR 15,0,0:LOCATE 11,14:PRINT "LASER
        BARRAGE":LOCATE 13,11:PRINT "HIT ANY KEY
        TO START"
AB 580 B=RND(N1):A$=INKEY$:IF A$="" THEN 580 ELSE
        RETURN
```

Rebound

Chris Metcalf and Marc Sugiyama

Here's a fast, smooth, all machine language adaptation of a classic arcade game. With the modifications included below, it runs on any IBM PC (color/graphics or monochrome adapter) with at least 64K RAM and a disk drive, or on any Enhanced Model PCjr.

"Rebound" takes advantage of machine language to streamline the action in this arcade-style game patterned after the popular *Breakout*. By controlling a paddle at the bottom of the screen, your goal is to knock out all the bricks at the top of the screen with a bouncing ball.

Unlike most action games, Rebound works on all three popular types of IBM personal computers. Program 1 is for an IBM PC with the color/graphics adapter. Program 2 consists of modifications to make Program 1 work on an IBM PC with the monochrome adapter. And although Program 1 works as is on an IBM PCjr, the modifications contained in Program 3 accelerate the game to compensate for Junior's slower execution speed.

Typing Instructions

If you have a color/graphics PC, type in Program 1. If you have a monochrome PC, type in Program 1 and substitute the lines in Program 2. If you have a PCjr, type in Program 1 and substitute the lines in Program 3. *To be safe, save the program on disk before running it for the first time.*

Next, insert a disk in drive A and type RUN. The BASIC program will create a machine language file on disk with the filename REBOUND.EXE (the drive may whir on and off a few times as the file is created).

When the Ok prompt reappears, exit BASIC to DOS by typing SYSTEM. Make sure the disk with the REBOUND.EXE file is in drive A. To run Rebound, type REBOUND.EXE at the DOS prompt. Almost instantly, the game screen will appear.

Eight Chances for Glory

To start playing Rebound, press the Enter key. The first ball starts moving downward from the middle of the screen. Your job is to keep it from falling off the bottom of the screen by bouncing it upward toward the rows of bricks.

To bounce the ball, move the paddle back and forth with the left and right Shift keys. You'll have to anticipate where the ball will bounce next, because the paddle can't always move across the screen as fast as the ball (otherwise, the game would be too easy). If you miss a ball, another one starts falling. You get a total of eight balls before the game ends in defeat. If you succeed in knocking out all the bricks, the program resets for another game.

There are five rows of bricks. Bricks on the lowest row are worth one point each, bricks on the second-lowest row are worth two points, and so on.

You can freeze Rebound at any time by pressing the space bar. Press any other key to resume play.

The Esc key restarts a game in progress by replacing all missing bricks and lost balls. It also resets the score to zero.

To stop the game entirely and exit to DOS, press Ctrl-Break on the PC or Function-Break (Fn-B) on the PCjr.

Program 1. Rebound for IBM PC (Color/Graphics)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
EL 10 ON ERROR GOTO 70
FB 20 OPEN "o",1,"rebound.exe"
CH 30 READ A : IF A<0 THEN 50
NE 40 PRINT #1,CHR$(A); : GOTO 30
EP 50 FOR I=1 TO -1*A : PRINT #1,CHR$(0); : NEXT
HJ 60 GOTO 30
QK 70 IF ERR <> 4 THEN ON ERROR GOTO 0
CB 80 CLOSE 1 : END
BF 100 DATA 77,90,32,1,4,0,1,0
KH 110 DATA 32,-3,255,255,74,0,128,0
ID 120 DATA 208,144,-4,32,-3,205,11,-2
PK 130 DATA 6,-479,30,184,-2,80,184,59
OK 140 DATA 0,142,216,142,192,176,3,180
DN 150 DATA 0,205,16,185,-2,182,24,178
OC 160 DATA 79,176,0,183,7,180,6,205
FP 170 DATA 16,181,32,180,1,205,16,182
FD 180 DATA 3,178,3,183,0,232,107,3
DB 190 DATA 176,218,179,15,232,107,3,178
BC 200 DATA 4,232,95,3,176,196,185,72
ME 210 DATA 0,232,97,3,178,76,232,82
```

```
LH 22Ø DATA 3,176,191,232,84,3,182,4
NA 23Ø DATA 178,3,232,7Ø,3,176,179,232
FI 24Ø DATA 72,3,178,76,232,6Ø,3,176
NN 25Ø DATA 179,232,62,3,254,198,128,254
CG 26Ø DATA 24,114,229,178,4,182,1,19Ø
DM 27Ø DATA 199,Ø,232,223,2,198,6,45
OD 28Ø DATA Ø,56,191,3,Ø,198,133,94
QL 29Ø DATA Ø,48,79,117,248,19Ø,26,Ø
BP 3ØØ DATA 232,179,2,198,6,1Ø,Ø,92
EH 31Ø DATA 198,6,6,Ø,128,198,6,8
NH 32Ø DATA -2,191,5,Ø,182,5,178,4
JF 33Ø DATA 183,Ø,19Ø,182,Ø,247,199,1
AF 34Ø DATA Ø,117,3,19Ø,189,Ø,185,9
OM 35Ø DATA Ø,232,159,2,226,251,232,113
OJ 36Ø DATA 2,79,117,224,128,62,11,-2
BL 37Ø DATA 117,18,198,6,11,Ø,1,232
NN 38Ø DATA 166,2,128,62,12,Ø,1,117
OB 39Ø DATA 3,233,1Ø6,1,254,14,45,Ø
BI 4ØØ DATA 128,62,45,Ø,48,115,3,233
FJ 41Ø DATA 145,1,19Ø,26,Ø,232,84,2
QP 42Ø DATA 182,24,178,Ø,19Ø,225,Ø,232
HB 43Ø DATA 96,2,198,6,9,Ø,35,18Ø
OC 44Ø DATA Ø,2Ø5,26,139,194,37,255,15
GB 45Ø DATA 178,2Ø,246,242,128,196,4,246
QO 46Ø DATA 196,1,116,3,128,196,52,138
IL 47Ø DATA 212,182,11,137,22,-2,198,6
DB 48Ø DATA 4,Ø,2,128,252,39,119,5
BN 49Ø DATA 198,6,4,Ø,3,18Ø,2,2Ø5
DH 5ØØ DATA 22,36,3,116,25,6Ø,3,116
IF 51Ø DATA 21,6Ø,1,116,2,176,255,2
MH 52Ø DATA 6,9,Ø,6Ø,2,124,7,6Ø
BG 53Ø DATA 68,127,3,162,9,Ø,232,46
IC 54Ø DATA 1,16Ø,6,Ø,44,127,177,1Ø
JL 55Ø DATA 246,225,247,216,5,16Ø,15,139
KC 56Ø DATA 2ØØ,226,254,16Ø,5,Ø,2,6
JP 57Ø DATA 6,Ø,162,5,Ø,114,3,233
QM 58Ø DATA 173,Ø,139,22,-2,16Ø,4,Ø
FI 59Ø DATA 36,1,2Ø8,224,44,1,2,2Ø8
QK 6ØØ DATA 16Ø,4,Ø,36,2,44,1,128
CB 61Ø DATA 54,7,Ø,1,117,2,176,Ø
PC 62Ø DATA 2,24Ø,128,254,24,119,39,183
CB 63Ø DATA Ø,232,3,2,18Ø,8,2Ø5,16
FB 64Ø DATA 6Ø,32,116,88,191,13,Ø,185
EK 65Ø DATA 4,Ø,252,242,174,117,41,232
AH 66Ø DATA 247,Ø,128,54,4,Ø,2,198
CP 67Ø DATA 6,7,-2,235,179,139,22,2
PH 68Ø DATA Ø,232,213,Ø,139,22,-2,232
KL 69Ø DATA 2Ø6,Ø,232,175,Ø,18Ø,14,176
EK 7ØØ DATA 7,2Ø5,16,233,2,255,191,17
LH 71Ø DATA Ø,185,3,Ø,242,174,117,5
```

```
BC 720 DATA 128,54,4,0,1,191,20,0
KA 730 DATA 185,6,0,242,174,116,3,233
JF 740 DATA 125,255,128,54,4,0,2,233
BA 750 DATA 117,255,128,54,8,0,1,117
JN 760 DATA 19,82,139,22,2,0,232,143
GK 770 DATA 0,90,232,124,0,137,22,-2
LH 780 DATA 235,9,144,135,22,-2,137,22
HB 790 DATA 2,0,180,1,205,22,116,39
MC 800 DATA 180,0,205,22,128,252,1,117
AC 810 DATA 8,198,6,11,-2,233,219,253
DK 820 DATA 60,32,117,7,180,0,205,22
KG 830 DATA 235,12,144,10,196,117,7,176
GG 840 DATA 3,180,0,205,16,203,128,62
CI 850 DATA 10,-2,116,3,233,215,254,190
PC 860 DATA 99,0,232,231,0,139,22,2
OD 870 DATA 0,232,56,0,139,22,-2,232
MF 880 DATA 49,0,232,18,0,232,7,1
NP 890 DATA 128,62,12,0,1,116,206,233
KG 900 DATA 151,253,190,128,0,235,218,182
GN 910 DATA 24,138,22,9,0,190,209,0
GF 920 DATA 232,206,0,195,82,183,0,232
FM 930 DATA 14,1,176,9,179,3,232,14
HN 940 DATA 1,90,195,82,183,0,232,255
JE 950 DATA 0,176,32,179,7,232,255,0
DD 960 DATA 90,195,254,14,10,0,83,81
LE 970 DATA 82,86,138,254,128,238,6,128
QM 980 DATA 230,1,208,230,128,234,4,42
ME 990 DATA 214,128,226,252,2,214,185,4
OP 1000 DATA 0,121,5,178,0,185,2,0
PE 1010 DATA 128,194,4,138,247,128,250,74
ED 1020 DATA 114,3,185,2,0,183,0,232
BE 1030 DATA 190,0,176,32,179,7,232,193
AF 1040 DATA 0,183,11,42,254,2,62,97
GO 1050 DATA 0,136,62,97,0,183,58,190
FK 1060 DATA 2,0,58,188,95,0,119,18
DF 1070 DATA 138,156,95,0,128,235,10,136
DM 1080 DATA 156,95,0,254,132,94,0,78
GN 1090 DATA 117,232,190,26,0,232,43,0
DH 1100 DATA 128,238,6,177,5,210,230,246
CP 1110 DATA 214,58,54,6,0,114,4,136
HA 1120 DATA 54,6,0,94,90,89,91,195
AC 1130 DATA 82,176,1,181,5,177,4,182
PL 1140 DATA 23,178,75,183,7,180,7,205
JH 1150 DATA 16,90,195,83,173,139,208,183
GL 1160 DATA 0,232,84,0,172,60,0,116
HJ 1170 DATA 6,180,14,205,16,235,245,91
CN 1180 DATA 195,81,86,183,0,232,64,0
DM 1190 DATA 172,60,0,116,15,138,200,181
MD 1200 DATA 0,172,138,216,172,232,58,0
DB 1210 DATA 2,209,235,233,94,89,195,232
```

```
DA 1220 DATA 19,0,190,161,0,232,195,255
IB 1230 DATA 180,0,205,22,10,196,116,18
FI 1240 DATA 128,252,28,117,243,139,22,161
CA 1250 DATA 0,178,4,190,229,0,232,192
LJ 1260 DATA 255,195,198,6,12,0,1,195
PD 1270 DATA 80,180,2,205,16,88,195,185
GO 1280 DATA 1,0,180,9,205,16,195,-19
EH 1290 DATA 219,219,219,219,179,218,191,196
BP 1300 DATA 205,218,191,213,184,4,1,66
PB 1310 DATA 97,108,108,115,32,114,101,109
PF 1320 DATA 97,105,110,105,110,103,58,32
JM 1330 DATA 0,32,32,32,32,32,32,32
NI 1340 DATA 32,32,32,32,32,82,32,69
GE 1350 DATA 32,66,32,79,32,85,32,78
JJ 1360 DATA 32,68,32,32,32,32,32,32
BI 1370 DATA 32,32,32,32,32,32,32,32
PF 1380 DATA 32,32,32,83,99,111,114,101
NL 1390 DATA 58,32,-4,27,2,67,111,110
EN 1400 DATA 103,114,97,116,117,108,97,116
HE 1410 DATA 105,111,110,115,33,32,32,89
LJ 1420 DATA 111,117,32,119,105,110,33,0
EO 1430 DATA 25,2,84,114,105,117,109,112
PA 1440 DATA 104,33,32,32,78,111,116,104
NJ 1450 DATA 105,110,103,32,99,97,110,32
LF 1460 DATA 115,116,111,112,32,109,101,33
GJ 1470 DATA 0,31,16,72,105,116,32,69
FD 1480 DATA 110,116,101,114,32,116,111,32
CD 1490 DATA 66,101,103,105,110,0,4,1
HM 1500 DATA 219,4,2,219,0,2,4,219
OC 1510 DATA 4,6,219,2,4,219,0,20
DJ 1520 DATA 12,32,24,15,32,26,12,32
OA 1530 DATA 0,1,9,32,1,9,213,6
AD 1540 DATA 9,205,1,9,184,1,9,32
BE 1550 DATA 0,79,9,32,0,72,7,32
NO 1560 DATA -8,83,84,65,67,75,32,32
NL 1570 DATA 32,83,84,65,67,75,32,32
NO 1580 DATA 32,83,84,65,67,75,32,32
NB 1590 DATA 32,83,84,65,67,75,32,32
MI 1600 DATA 32,83,84,65,67,75,32,32
ML 1610 DATA 32,83,84,65,67,75,32,32
MO 1620 DATA 32,83,84,65,67,75,32,32
MB 1630 DATA 32,83,84,65,67,75,32,32
ME 1640 DATA 32,83,84,65,67,75,32,32
MH 1650 DATA 32,83,84,65,67,75,32,32
NK 1660 DATA 32,83,84,65,67,75,32,32
NN 1670 DATA 32,83,84,65,67,75,32,32
NA 1680 DATA 32,83,84,65,67,75,32,32
ND 1690 DATA 32,83,84,65,67,75,32,32
MK 1700 DATA 32,83,84,65,67,75,32,32
MN 1710 DATA 32,83,84,65,67,75,32,32
KG 1720 DATA 32
```

Program 2. Rebound Modifications for Monochrome PC

```
OM 1290 DATA 176,177,178,219,179,218,191,196
GC 1490 DATA 66,101,103,105,110,0,4,15
NB 1500 DATA 176,4,7,177,0,2,15,178
FE 1510 DATA 4,7,219,2,15,178,0,20
IG 1520 DATA 7,32,24,15,32,26,7,32
JG 1530 DATA 0,1,7,32,1,7,213,6
JB 1540 DATA 7,205,1,7,184,1,7,32
PE 1550 DATA 0,79,7,32,0,72,7,32
```

Program 3. Rebound Modifications for PCjr

```
CN 540 DATA 1,160,6,0,44,127,177,6
BP 550 DATA 246,225,247,216,5,206,9,139
```

Quickversi

David Bohlke

"Quickversi" is a version of the popular board game reversi which pits you against the computer. The program runs on any PCjr or a PC with BASICA and the color/graphics adapter.

In this version of reversi you'll be competing against the superbrain of your IBM computer. But don't worry—the computer plays only well enough to win against beginning or intermediate players. If you are a beginner, in time you should be able to develop a winning strategy to consistently beat the computer.

Although this program isn't the smartest reversi player around, I've added an extra timing dimension to make the game more challenging for experienced players. Hence the name "Quickversi." The computer keeps track of how long it takes you to make a move and deducts one point for each second. Thus, you can compete as usual to capture the most pieces, but you'll also need to play very quickly to finish with a good score.

How to Play Reversi

If you're new to this game, it's quite easy to learn—even easier than checkers. Here's how it works.

When you run the program, an 8 × 8 grid covers most of the screen. Alongside this board are various indicators which display the total time (in seconds) used by both the player and the IBM to make their moves; the player's current score; and a vertical bar chart which compares the number of pieces held by the player and the computer. Prompts appear at the bottom of the screen.

When the game starts, both you and the computer have two pieces grouped at the center of the board. The program randomly determines who moves first. Then you and the computer take turns putting new pieces on the board in order to capture opposing pieces. To put a new piece on the board, you use the cursor keys to move a flashing white cursor to an empty square, then press Enter. To capture enemy pieces, you

have to enclose them on two sides (either vertically, horizontally, or diagonally) with your own pieces.

Capturing an opposing piece does not remove it from the board. Instead, it converts the piece to *your* side. The bar chart on the right will indicate how many pieces each player has.

It's fairly typical to have more than one possible move during each turn. It's up to you to decide which move is best. However, note that moves which do not capture at least one opposing piece are *not* legal. In other words, every move must capture an enemy piece. If no legal moves are available, you must pass by pressing the Esc (Escape) key at the upper-left corner of the keyboard. If you try to make an illegal move, the screen will flash briefly and you will be prompted to try again.

Advanced Scoring

If you're new to reversi, you can start out by merely competing for the greatest number of pieces. But when you become a more advanced player, you can test your skill against the computer in a game of quick moves.

The score indicators above the Quickversi board are for experienced players. The computer keeps track of how long it takes you to make a move and subtracts it from your overall score. Also, the difference between the number of pieces you have and the number of pieces the computer has will be multiplied by ten. If the computer's pieces outnumber yours, the result will be a negative number. The difference between the number of seconds it takes for the computer to make a move and the time it takes for you to move also is considered.

For instance, if a game ends and you have 40 pieces plus 400 seconds in move time, and the computer has 24 pieces and 300 seconds in move time, your final score would be

(40−24)*10 + (300−400) = 60 points

Obviously, you have to play Quickversi very fast and still manage to capture a large share of pieces to wind up with a decent score. Most likely, you'll tally a negative score. A score of more than 500 qualifies you as a reversi pro.

How It Works

For those who are interested in BASIC programming, here's a breakdown of Quickversi:

Line	Description
5–99	Initialization
30–32	Flip directions
50–60	Grid lines
70–76	Initial grid occupation
80–84	Value for grid locations
290	First player to move
300–392	Human's move
300–320	Get key input
360	Check if legal move
370	Display flips
380–382	Flip pieces
400–492	Computer's move
410	Check each square
420	Check empty squares with higher point values
421–422	Move cursor
430	Check if legal move
440–442	Use square with highest point value
450	No move
460–470	Squares to flip
480	Flip squares
800–808	Legal move at square?
820–838	Put pieces to flip in F(20)
850–880	Display score bars
900–902	Color in square
910–912	Color in cursor
930	Print score
990–996	End of game prompt

Variable List

B(99)	Player pieces
P(99)	Point value for squares
F(20)	Possible flips to be made
D(8)	Direction of flips
FNTIM	Timer function
T	Timer
CT	Computer time
HT	Human time
S	Square grid location
CS	Number of computer pieces
HS	Number of human pieces
SQ	Square to move to
PF	Piece to move flag
PM	Piece to flip flag
B$,C$	Play music strings

C H A P T E R 2

Quickversi

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
QL 5 DEFINT A-Z:KEY OFF
CM 7 DEF SEG=0:DEF FNTIM =(PEEK(1132)+PEEK(1133)*
     256)/18.2:RANDOMIZE FNTIM
KO 10 CLS:SCREEN 1:COLOR 6,1
AG 20 DIM B(99),P(99),F(20),D(8)
FD 22 A$="QUICKVERSI":FOR I=1 TO 11:LOCATE I+6,1:
      PRINT MID$(A$,I,1);:NEXT
IG 30 FOR I=1 TO 8:READ D(I):NEXT
HE 32 DATA -11,-10,-9,-1,1,9,10,11
OP 50 FOR I=10 TO 250 STEP 30:LINE (I,10)-(I+1,17
      0),3,BF:NEXT
KD 60 FOR I=10 TO 170 STEP 20:LINE (10,I)-(250,I+
      1),3,BF:NEXT I
FH 70 FOR I=0 TO 9:B(I)=9:B(I+90)=9:NEXT:FOR I=1
      TO 8:B(I*10)=9:B(I*10+9)=9:NEXT I
FP 72 B(44)=1:B(45)=2:B(54)=2:B(55)=1
PN 75 S=44:GOSUB 900:S=55:GOSUB 900
OA 76 S=45:GOSUB 900:S=54:GOSUB 900
PO 80 FOR I=1 TO 4:FOR J=1 TO 8:READ X:K=I*10+J:P
      (K)=X:P((4-I)*20+K+10)=X:NEXT:NEXT
HO 81 DATA 9,2,8,6,6,8,2,9
JA 82 DATA 2,1,3,4,4,3,1,2
DC 83 DATA 8,3,7,5,5,7,3,8
DE 84 DATA 6,4,5,0,0,5,4,6
PH 90 CT=0:HT=0:GOSUB 930
FP 99 LOCATE 1,34:PRINT "IBM You";:GOSUB 850
NL 290 RANDOMIZE FNTIM:IF RND<.5 THEN 400
KK 300 S=44:T=FNTIM
EC 310 LOCATE 23,2:PRINT"Position cursor, press E
       nter";
FN 312 LOCATE 24,4:PRINT CHR$(24);" ";CHR$(25);"
       ";CHR$(26);" ";CHR$(27);" or ESC if no mov
       e ";
CF 320 HT=HT+(FNTIM-T):T=FNTIM:GOSUB 930
CN 321 A$=INKEY$:IF A$="" THEN 330
CP 322 IF ASC(A$)=13 THEN 360 ELSE IF ASC(A$)=27
       THEN 390
KG 323 K=0:IF LEN(A$)>1 THEN K=ASC(MID$(A$,2))
QM 325 IF K=72 THEN S=S-10:IF S<11 THEN S=S+80
CG 326 IF K=80 THEN S=S+10:IF S>90 THEN S=S-80
FA 327 IF K=77 THEN S=S+1:IF B(S)=9 THEN S=S-8
BD 328 IF K=75 THEN S=S-1:IF B(S)=9 THEN S=S+8
LM 330 C=3:GOSUB 910:GOSUB 900:GOTO 320
KM 360 IF B(S)<>0 THEN 330
PH 361 PM=2:PF=1:SM=0:GOSUB 800:IF SM<>0 THEN 370
DF 362 COLOR 1,0:PLAY "MF O1 CAE":COLOR 6,1:GOTO
       320
```

```
KC 370 GOSUB 900:N=1:F(1)=S:C=PM:PLAY "A":GOSUB 9
       10:GOSUB 820
LM 380 FOR K=1 TO N:B(F(K))=PM:S=F(K):GOSUB 910:N
       EXT:GOSUB 850
AC 382 FOR K=1 TO N:S=F(K):PLAY "MF O3 AD":GOSUB
       900:NEXT
OD 390 HT=HT+INT(FNTIM-T):GOSUB 930
HC 392 IF CS+HS=64 THEN 990
NJ 400 SQ=0:PF=2:PM=1:T=FNTIM
EL 401 LINE (0,171)-(318,199),0,BF
CP 402 LOCATE 23,5:PRINT"IBM's move...";
HM 410 C=2:FOR S=11 TO 88
PK 420 IF P(S)<P(SQ) OR B(S)<>0 THEN 450
GC 421 C=1:GOSUB 910
FN 422 PLAY "MB L36 O0 F":GOSUB 900
EI 430 SM=0:GOSUB 800
IA 440 IF P(SM)=P(SQ) AND RND>.7 THEN SQ=SM
LO 442 IF P(SM)>P(SQ) THEN SQ=SM
FA 450 NEXT S:IF SQ=0 THEN COLOR 0,0:PLAY "MF O3
       ABCDEF":COLOR 6,1:GOTO 300
OF 460 S=SQ:F(1)=SQ:N=1:GOSUB 820
LH 470 C=1:FOR K=1 TO N:B(F(K))=PM:S=F(K):GOSUB 9
       10:NEXT:GOSUB 850
CB 480 FOR K=1 TO N:S=F(K):PLAY "MF O2 FA":GOSUB
       900:NEXT
JE 490 CT=CT+FNTIM-T:GOSUB 930
KL 492 IF CS+HS=64 THEN 990 ELSE 300
MH 800 FOR J=1 TO 8:K=S+D(J):IF B(K)<>PF THEN 808
MF 802 K=K+D(J):IF B(K)=PF THEN 802
IC 804 IF B(K)<>PM THEN 808
PO 806 SM=S:RETURN
HA 808 NEXT:RETURN
FC 820 FOR J=1 TO 8:K=S+D(J):IF B(K)<>PF THEN 838
AH 822 K=K+D(J):IF B(K)=PF THEN 822
NL 824 IF B(K)<>PM THEN 838
CD 825 K=K-D(J)
DG 828 IF K=S THEN 838
NH 830 N=N+1:F(N)=K:GOTO 825
HG 838 NEXT:RETURN
GE 850 CS=0:HS=0:LINE (260,10)-(315,199),0,BF
GA 851 FOR S=11 TO 88
JA 852 IF B(S)=1 THEN 870 ELSE IF B(S)=2 THEN 860
        ELSE 880
PE 860 SOUND S*10,.5:HS=HS+1:LINE (290,30)-(304,H
       S*3+30),2,BF:GOTO 880
JI 870 SOUND S*20,.2:CS=CS+1:LINE (265,30)-(275,C
       S*3+30),1,BF
DM 880 LOCATE 3,33:PRINT CS;:LOCATE 3,36:PRINT HS
       ;:NEXT:RETURN
```

```
IG 9ØØ X=(S-INT(S/1Ø)*1Ø)*3Ø:Y=INT(S/1Ø)*2Ø
NM 9Ø2 LINE (X-18,Y-8)-(X+8,Y+9),B(S),BF:RETURN
JI 91Ø X=(S-INT(S/1Ø)*1Ø)*3Ø:Y=INT(S/1Ø)*2Ø
HJ 912 LINE (X-1Ø,Y-4)-(X,Y+4),C,BF:RETURN
LG 93Ø LOCATE 1,2:PRINT "IBM=";INT(CT);" You=";IN
       T(HT);" Score=";INT(HS-CS)*1Ø+INT(CT-HT);"
        ";:RETURN
FP 99Ø LINE (Ø,171)-(318,199),Ø,BF
JF 991 LOCATE 24,4:PRINT"Press <Enter> for new ga
       me ";
EK 995 A$=INKEY$:IF A$="" THEN 995
DE 996 IF ASC(A$)=13 THEN RUN ELSE 995
```

Bowling Champ

Joseph Ganci
Translation by Tim Victor

Now you can go bowling without the expense of renting special shoes or suffering the embarrassment of rolling a gutter ball in front of dozens of people. "Bowling Champ" is a game for one to four players which runs on any IBM PC or PCjr.

Some games, such as *Pac-Man* or *Adventure*, create their own unique fantasy worlds, while others are simulations of reality. "Bowling Champ" is an example of the latter.

It's not easy to take a game with countless physical variables (such as bowling) and reduce it to numbers so that it can be recreated by a computer—especially a microcomputer. Compromises must be made. Usually, the game must be modified in major ways to make it possible to program. The result is a hybrid game, an approximation of reality, that resembles the original but has new aspects of its own.

Bowling Champ is a reasonable simulation of a game of tenpins, given the limitations imposed by a BASIC program which must remain short enough to publish. The elements of skill and luck have been preserved, and the scoring is authentic.

Up to Four Players

When you first run Bowling Champ, the program asks for the number of players. Up to four people can play.

Next, you enter the players' names. To fit the names on the screen in 40-column format, the program truncates names to eight characters.

Now you're ready to bowl the first frame. The bowling ball rapidly moves up and down across the alley until you press the space bar. This rolls the ball down the alley and knocks over the pins—unless you've thrown a gutter ball. The trick is to time your release so the ball rolls down the center of the alley to score a strike.

In case you're unfamiliar with how a game of tenpins is scored, here's a brief summary:

A game consists of ten frames, or turns. Each player gets one or two balls per frame. If you roll a strike—knocking down all ten pins with the first ball—you don't get a second ball, but the current ball's score is ten plus the total of your next two throws.

If some pins are left standing after your first ball, you get a second ball. If you knock down all the remaining pins, it counts as a spare, and the current ball's score is ten plus your next throw.

If any pins remain after your second ball (no strike or spare), the number of pins knocked down in that frame is added to your previous score.

Rolling a spare in the tenth (last) frame gains you one extra ball; rolling a strike in the tenth frame gains two extra balls.

Therefore, a perfect game—ten strikes during regular play plus two strikes with the extra bowling balls—scores 300 points. Needless to say, this doesn't happen very often, either in real bowling or in Bowling Champ.

Since Bowling Champ follows every scoring rule for regular bowling, you can learn how to score by carefully observing the game. The only difference is that the computer doesn't wait until the end of a frame to update the score; it updates it after every ball.

Adjusting the Difficulty

Some new players or young people may find that the ball moves too fast for them to aim. On the other hand, more experienced players may want to speed up the ball to make the game harder. You can easily make either modification by changing the delay loop at the beginning of line 530.

The first statement in line 530 is FOR I=1 TO 10:NEXT. Replacing the 10 with a larger number slows down the ball, and substituting a smaller number speeds up the ball. For youngsters, you might try a value of 20 or 30. For expert players, remove the delay loop altogether so that line 530 reads IF V=24 OR V=16 THEN D=−D.

You'll also find that Bowling Champ runs slightly faster on a PC, especially with the IBM Monochrome Display, than on a PCjr.

CHAPTER 2

Bowling Champ

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
DL 1Ø DIM NA$(3),S(3),T(3) : COLOR 3,Ø,Ø
BB 2Ø WIDTH 4Ø : KEY OFF : LOCATE 1,1,Ø : CLS : D
      EF SEG=Ø : POKE 1Ø47,64
EC 3Ø LOCATE 8,12 : PRINT "BOWLING CHAMP!"
PA 4Ø LOCATE 13,7 : PRINT "How many bowlers? (1-4
      ):"
MN 5Ø A$="" : WHILE A$="" : A$=INKEY$ : WEND
DG 6Ø IF ASC(A$)<49 OR ASC(A$)>52 THEN 5Ø ELSE A=
      VAL(A$)
II 7Ø FOR I=1 TO A : LOCATE 15+I,8 : PRINT "Bowle
      r";I;
FL 8Ø LOCATE ,16 : PRINT"'s name:";
JN 9Ø INPUT A$ : NA$(I-1)=LEFT$(A$,8) : NEXT
DD 1ØØ '     --> DRAW GAME SCREEN <--
AB 11Ø CLS : COLOR 15 : LOCATE 1,1Ø : PRINT "1  2
         3  4  5  6  7  8  9  1Ø"
NI 12Ø LOCATE 2,1Ø : FOR J=1 TO 31 : PRINT CHR$(1
       96);: NEXT
LH 13Ø FOR I=1 TO A : LOCATE 2*I+1,1 : COLOR I+1
       : PRINT NA$(I-1);: LOCATE ,12
AC 14Ø COLOR 15 : FOR J=12 TO 36 STEP 3 : PRINT C
       HR$(179)+"  ";: NEXT
PK 15Ø LOCATE 2*I+2,1Ø : FOR J=1 TO 31 : PRINT CH
       R$(196); : NEXT : NEXT
JL 16Ø LOCATE 12,1 : FOR I=Ø TO A-1 : IF I=2 THEN
        LOCATE 14,1
CA 17Ø COLOR I+2 : PRINT NA$(I);":";SPC(19-LEN(NA
       $(I))); : NEXT : COLOR 15
OH 18Ø LOCATE 15,1 : FOR I=1 TO 39 : PRINT CHR$(1
       96);: NEXT
PF 19Ø LOCATE 25,1 : FOR I=1 TO 39 : PRINT CHR$(1
       96);: NEXT
BN 2ØØ ' --> INITIALIZE SCORE STATE <--
HO 21Ø FOR I=Ø TO A-1 : S(I)=1 : NEXT
PO 22Ø '         --> MAIN LOOP <--
QI 23Ø FOR Q=1 TO 1Ø : FOR Z9=Ø TO A-1
QJ 24Ø COLOR,Z9+2 : FOR I=16 TO 24 : LOCATE I,1 :
        PRINT SPC(39) : NEXT
MI 25Ø B1=Ø : GOSUB 39Ø
EF 26Ø IF J1<>1Ø THEN B1=1 : GOSUB 43Ø
FD 27Ø IF Q=1Ø THEN ON S GOTO 28Ø,31Ø,31Ø,28Ø,34Ø
NA 28Ø NEXT : NEXT : LOCATE 16,1Ø : PRINT "PLAY A
       GAIN?(Y OR N)"
JO 29Ø A$="" : WHILE A$="" : A$=INKEY$ : WEND : I
       F A$="Y" THEN RUN ELSE COLOR 7,Ø : END
IH 3ØØ ' --> 1ØTH FRAME : EXTRA BALLS <--
```

```
IK 31Ø LOCATE 2Ø,1 : PRINT "TAKE TWO MORE BALLS,
       ";NA$(Z9);
PB 32Ø FOR I=1 TO 1ØØØ : NEXT : LOCATE ,1 : PRINT
        SPC(29)
DP 33Ø S(Z9)=S-1 : B1=1 : GOSUB 39Ø : IF J<>1Ø TH
       EN 37Ø ELSE 36Ø
MM 34Ø LOCATE 2Ø,1 : PRINT "TAKE ONE MORE BALL, "
       ;NA$(Z9);
MC 35Ø FOR I=1 TO 1ØØØ : NEXT : LOCATE ,1 : PRINT
        SPC(28)
DM 36Ø S(Z9)=1 : B1=2 : GOSUB 39Ø : GOTO 28Ø
KM 37Ø S(Z9)=1 : B1=2 : GOSUB 43Ø : GOTO 28Ø
IN 38Ø '       --> FIRST BALL <--
GM 39Ø COLOR 15 : LOCATE 17,39
GO 4ØØ FOR I=1 TO 31 : READ PC : PRINT CHR$(PC);
       : NEXT : RESTORE
NA 41Ø PS=1=1 : J1=Ø : GOTO 44Ø
LB 42Ø '       --> SECOND BALL <--
FH 43Ø PS=Ø
LB 44Ø GOSUB 5ØØ : T=T(Z9) : S=S(Z9) : T=T+J
FA 45Ø ON S(Z9) GOSUB 68Ø,7ØØ,72Ø,74Ø,76Ø
FM 46Ø T(Z9)=T : S(Z9)=S : COLOR Z9+2,Ø
JF 47Ø LOCATE 14+(Z9<2)*2,31+(Z9/2=INT(Z9/2))*2Ø
       : PRINT T(Z9)
NO 48Ø COLOR Ø,Z9+2 : RETURN
JG 49Ø '         --> ROLL BALL <--
CC 5ØØ H=1 : V=24 : D=-1 : COLOR Ø,Z9+2
DI 51Ø WHILE INKEY$="" : LOCATE V,H : PRINT " ";
GK 52Ø V=V+D : LOCATE V,H : PRINT "O";
FD 53Ø FOR I=1 TO 1Ø : NEXT : IF V=24 OR V=16 THE
       N D=-D
LO 54Ø WEND : FOR H=2 TO 35 : LOCATE V,H-1 : PRIN
       T " O";: A$=INKEY$
JM 545 IF A$="J" THEN WHILE INKEY$="" : WEND
EB 55Ø SOUND 37,.5 : SOUND 32767,.1 : NEXT
NM 56Ø J=Ø : WHILE H<4Ø
AG 57Ø IF SCREEN(V,H)=234 THEN J=J+1 ELSE 61Ø
ME 58Ø FOR D=-1 TO 1 STEP 2 : X1=V : X2=H
BK 59Ø X1=X1+D : X2=X2+1 : IF SCREEN(X1,X2)=234 T
       HEN LOCATE X1,X2 : PRINT " ";: J=J+1 : SOU
       ND 74,.5 : SOUND 32767,.1 : GOTO 59Ø
NC 6ØØ NEXT
OM 61Ø LOCATE V,H-1 : PRINT " O";: H=H+1 : WEND
NC 62Ø J1=J1+J
GL 63Ø LOCATE 2*Z9+3,7+3*Q+B1 : COLOR -(2+Z9)*(B1
       =Ø),-(2+Z9)*(B1<>Ø) : G=J+48
HJ 64Ø IF J1=1Ø THEN IF PS THEN G=88 ELSE G=47
CC 65Ø PRINT CHR$(G) : COLOR Ø,2+Z9
FJ 66Ø LOCATE V,H-1 : PRINT " ";: RETURN
```

```
MK 670 '      --> SCORING ROUTINES <--
IA 680 IF J1=10 THEN IF PS THEN S=2 ELSE S=5
NC 690 RETURN
NF 700 T=T+J : IF J=10 THEN S=3 ELSE S=4
MD 710 RETURN
MA 720 T=T+J*2 : IF J<>10 THEN S=4
MH 730 RETURN
OD 740 T=T+J : IF J1=10 THEN S=5 ELSE S=1
NL 750 RETURN
FG 760 T=T+J : IF J=10 THEN S=2 ELSE S=1
NP 770 RETURN
GJ 1000 '      --> PIN DATA <--
OC 1001 DATA 234,31,29,29,234,31,29,29,234,28
II 1002 DATA 234,31,29,29,29,29,234,28,234,31
IJ 1003 DATA 29,29,234,28,234,31,29,29,234,31,234
```

Chess

John Krause

IBM "Chess" is written in two parts with the intelligence routines written entirely in machine language. Additional features make it a powerful chess program. It has multiple skill levels, checking for illegal moves, one- and two-player modes, reverse moving, and many other features. The program requires a PC with at least 128K RAM, color/graphics adapter, BASICA, and a disk drive, or an Enhanced Model PCjr with Cartridge BASIC.

A computer chess game is great for those who can't always find a human opponent. But "Chess" is more than just a substitute for a live player. You might call it a "chess processor." It processes chess positions as easily as a word processor manipulates text. It contains all the features a chess player could ever want. Its thinking routines are written entirely in machine language for greater speed, and they use basic principles of artificial intelligence to simulate an actual human chess player.

Chess consists of two programs. First, type in and save each program. Then load and run Program 1. You'll have to wait about 15 seconds while it creates a BLOAD file on the disk called CHESS.BLD which contains the machine language. Once this file is created, Program 1 is no longer used. From now on, to play Chess, simply load and run Program 2.

After running Chess, you'll see a title screen for a few seconds while the computer prepares itself. Then the board is displayed with the pieces in their starting positions. You're in command of the white pieces versus the computer's black pieces on skill level 1, the easiest level. You should see a frame around the square in the lower-left corner of the board. This is the cursor which takes the place of your hand for moving and capturing pieces.

Use the cursor keys to move the frame cursor atop the piece you wish to move. Press and release the Enter key. Now move the cursor to the square on which you want to place the piece and hit Enter again. Your piece moves to the new

square, and the computer responds instantly with a countermove.

Sorry, No Cheating

One of the most valuable features of IBM Chess is that it checks for illegal moves. If you try to make an illegal move, the computer buzzes and keeps your piece on its square. This feature is not perfect, however. It won't catch illegal moves involving castling or *en passant* captures. But it will catch 99 percent of all illegal moves, including those that put your king in check, as well as the more obvious ones such as moving a pawn backward. If the computer accepts your move, it's probably legal, but not necessarily so. If the computer rejects your move, however, you can be sure that it is illegal.

If you're a beginner at chess, you'll find the move-checking feature especially valuable. Just by trying various moves and noting which ones the computer accepts, you can get a good idea of the way each piece can move.

Information about the current game is displayed at the top of the screen. *Move#* indicates the number of the move currently being made, counting from the start of the game. In chess, a move by both sides is considered one move. So, the move number is changed only after both sides have moved.

To Move indicates which side has the move. W means it is white's turn, and B means it is black's.

Normally, after you move, the computer automatically makes the next move. You can turn this off by pressing the T key to switch to two-player mode. Now you can play against another person with the computer acting as referee to check for illegal moves. To switch back to one-player mode, press T again.

You can also let the computer make moves for you by pressing the M key. The side that the computer plays depends on whose turn it is. By repeatedly pressing M, you can watch the computer play itself. (IBM Chess requires that all commands be in lowercase mode. Therefore, if you get no response, try pressing the Caps Lock key once and then press the desired letter again.)

Five Skill Levels

One of the advantages of a computer opponent over a human is that you can tell the computer exactly how hard you want it

to try to beat you, and it obediently plays at that level of difficulty. This is important because it's no fun if you always lose or always win effortlessly.

Level shows the current skill level from 1 to 5. You can change the level at any time by pressing keys 1–5. The difference between levels is the number of moves ahead that the computer looks. On level 1, for example, it looks ahead one full move or two half-moves (its move and your reply). Each succeeding level looks ahead one more half-move than the previous level.

Alas, the smarter play on the higher levels doesn't come without a price. The further ahead the computer looks, the more moves it must examine and, hence, the longer it thinks. Here's a rundown of the five levels:

Level 1: Beginner. Thinking time: one second. Lookahead: two half-moves. Fast but dumb.

Level 2: Intermediate. Thinking time: five seconds. Lookahead: three half-moves. Provides a reasonable challenge for impatient players.

Level 3: Tournament. Thinking time: two minutes. Lookahead: four half-moves. Since the usual time limit for tournament play is 40 moves in two hours, an average of three minutes per move, this level is best suited for serious players.

Level 4: Mate in two. Thinking time: 20 minutes. Lookahead: five half-moves. Capable of solving most mate-in-two problems.

Level 5: Postal chess. Thinking time: two hours. Lookahead: six half-moves. Simulates chess by mail where there is no time limit. Can avoid checkmate in two moves.

These thinking times are averages. The actual thinking time varies greatly depending on the position. For example, level 5 takes only five seconds with just two kings on the board. Also, these times are for the PC only. Since the PCjr runs at about two-thirds the speed of the PC, the thinking times for the PCjr are greater than the values shown above.

A Spectacular Blunder

It happens to everyone. It's inevitable. You've played for an hour, somehow managing to maneuver into a superior position in what you consider to be the best game of your life, only to throw it all away in a single, spectacular blunder.

Don't panic. You can take back the last half-move by pressing the B key (remember, you must be in lowercase mode). If you're in one-player mode, you need to press B again to take back your move and the computer's reply. In fact, you can press B repeatedly to take back several moves until you reach the starting position. This is possible because the computer records every move made in the game.

Another use for this feature is to allow the computer to suggest a move for you. If you don't have a good idea of where to move next, press M and the computer will move for you. If you like that move, press M again to continue with the computer's next move. But if you think you've found a better move, press B to take back the suggested move and make your own move.

Pressing the F key does the opposite of B. It moves forward through the move list up to the most advanced position. Note that every time a new move is made, the resulting position becomes the most advanced. So if you use B to backtrack to a previous position, and then make a new move, all subsequent stored moves are erased because they are no longer relevant.

If you have a printer, you can print the move list by pressing the P key. The list appears in three columns: the move numbers, white's moves, and black's moves. Each move is indicated by the square the piece moved from followed by the square it moved to. Each square is specified by its coordinates according to the numbers along the left side of the board and letters along the bottom.

You can also dump the screen image to the printer to get a hardcopy of a particularly interesting position. Before loading BASIC from DOS, type GRAPHICS with the DOS master disk in the drive. Then run Chess and press Shift-PrtSc (Fn-PrtSc on the PCjr) whenever you want to print the position.

Checkmate

The computer thinks by analyzing thousands of possible moves and countermoves and choosing what it considers to be the best move based on the relative value of the pieces. Most positions don't have just one best move, but several which are equally good, in which case the computer chooses among them at random. This random factor insures that every game will be different, and makes for varied and interesting play.

The computer announces checkmate when it occurs. However, there are a few quirks in the way the computer evaluates a checkmate. On levels 3–5, it announces checkmate prematurely. When this happens, the computer has determined that it's impossible to avoid checkmate on the *next* move or two—assuming both sides make the best moves.

Also, the computer doesn't know the subtle difference between checkmate and stalemate. Consequently, when a game is stalemated, the computer announces checkmate even though the game is a draw. Since the computer tries as hard as it can to checkmate its opponent, it also tries to achieve stalemate, possibly forcing a draw when it could have won. Fortunately, this rarely happens, because a stalemate requires unusual circumstances, such as when one side has only the king remaining.

You can start a new game at any time by pressing the N key. This sets up the pieces in the starting position with white on the bottom. If you want to play the black pieces, you can press the I key to invert the board, so you still play from the bottom. As with the N command, the board is reset to the starting position. However, the N and I commands retain the move list from the previous game. This allows you to replay the game using the F command. When replaying a game, be sure to reset the board by pressing I if the game was played in the inverted mode, or N if normal mode was used.

Set Up Any Position

You don't have to begin a game from the starting position. You can set up any position and begin playing from that point. If you want, you can first clear the board by pressing the C key. To add a piece or change a piece to a different one, move the cursor to the appropriate square, hold down either Shift or Ctrl, and press P, N, B, R, Q, or K for pawn, knight, bishop, rook, queen, or king, respectively. Holding down Shift adds one of the lower player's pieces, and Ctrl adds one of the upper player's pieces. (Just remember that Ctrl is above Shift on the keyboard.) A piece can be removed from the board by pressing the space bar. Note that these changes are not stored in the move list.

These commands allow you to experiment with hypothetical or downright ridiculous positions. The position doesn't even have to be legal. Live out your fantasy by giving yourself

ten queens versus the computer's lone king. Or invent your own type of chess by giving each side two kings, for example (although in this case the computer might get confused trying to determine a checkmate).

You can also set up a problem for the computer to solve, such as the mate-in-two problems published in many newspapers. To solve a mate-in-two problem, press C to clear the board, set up the position, press 4 to select level 4, and press M to start the computer thinking. After several minutes of deep thought, the computer will make a move (the solution) and announce checkmate. The only mate-in-two problems that the computer cannot solve are those which involve castling, *en passant* captures, or pawn promotion.

Special Moves

The computer never castles or captures *en passant* because, due to their complexity, these moves are not included in its thinking routine. But *you* can make these special moves. To castle, move the king *two* squares to the left or right. The rook moves automatically. To capture *en passant*, move your pawn diagonally to the proper square. The opponent's pawn is removed automatically. Remember, the computer doesn't check for illegal moves involving castling or *en passant* captures, so if you're a beginner, you should familiarize yourself with the rules on these special moves.

When a pawn reaches the opposite side of the board, it's automatically promoted to a queen. In the rare event that you would rather promote to a knight, bishop, or rook, you can easily make the change by positioning the cursor over the new queen and pressing N, B, or R with Shift or Ctrl. Note, however, that underpromotions are not stored in the move list.

Saving a Game

If you want to stop the present game and continue later, you can save the game on disk (in drive A) by pressing the S key. You'll see the prompt *Save:*. Type in a filename for your game and press Enter. The filename can be up to eight characters long. Don't type an extender; .CHS is added automatically. If a file on the disk already has the same name, it will be replaced.

To load a previously saved game, press the L key. Answer the *Load:* prompt with the filename and press Enter. (Don't

type the .CHS extender.) The L command restores the game exactly as it was when it was saved. Not only the position is restored, but also the move list and even the position of the cursor.

If the computer is unable to save or load a game, an error number is displayed. See Appendix A of the *BASIC Reference Manual* for a description of the error.

Besides allowing you to continue a game at a later time, the S and L commands can be used to create a library of your best games. To do this, press N or I just before saving. The game will come up in the starting position when loaded and can be replayed using the F command.

Chess Commands

B	Move backward
C	Clear board
F	Move forward
I	New game (inverted)
L	Load game
M	Computer's move
N	New game
P	Print move list
S	Save game
T	Two players
1–5	Level
Cursor Keys	Move cursor
Enter	Your move
Space Bar	Remove piece
Shift-P	Lower player's pawn
Shift-N	Lower player's knight
Shift-B	Lower player's bishop
Shift-R	Lower player's rook
Shift-Q	Lower player's queen
Shift-K	Lower player's king
Ctrl-P	Upper player's pawn
Ctrl-N	Upper player's knight
Ctrl-B	Upper player's bishop
Ctrl-R	Upper player's rook
Ctrl-Q	Upper player's queen
Ctrl-K	Upper player's king

Program 1. Chess (Machine Language)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
IG 1Ø DEF SEG=&HFFFF:IF PEEK(14)=253 THEN DEF SEG
      =&H17ØØ:GOTO 3Ø
IE 2Ø DEF SEG=&H1CØØ
EE 3Ø FOR I=1 TO 31:READ A$:FOR J=1 TO 143 STEP 2
BB 4Ø POKE K,VAL("&h"+MID$(A$,J,2)):K=K+1:IF K<82
      5 THEN NEXT:NEXT
KC 5Ø BSAVE"chess.bld",Ø,825
JG 6Ø DATA 1EB8311C8ED88C16E1ØØ8926E3ØØB84Ø1C8EDØ
      BCØØØ1E8ØAØØ8E16E1ØØ8B26E3ØØ1FCBFAB9Ø8ØØ8BD
      9C6875EØØCØE2F7C6Ø65EØØØØC6Ø6EØØØØØB8ØØØØBF
      FFFFE9Ø8Ø18A854CØØØ2
OJ 7Ø DATA 8554ØØ5Ø8AD88A8767ØØ8A9D4CØØ888767ØØ8A
      852CØØ5B888767ØØØ4Ø65Ø8B1E29ØØC6876ØØØØØ5B8
      A8F1ØØØ2A8D6ØØØC6856ØØØCØ83FFØØ75523AØE5FØØ
      7C4B7511BØØØE643E44Ø
IB 8Ø DATA E44Ø3AØ65EØØ723BA25EØØ8Ø3EDFØØØØ741DAØ
      4CØØ3AØ65CØØ7528Ø2Ø654ØØ3AØ65DØØ751E8ØF9E57
      E19FEØ6EØØØC388ØE5FØØ8AØE4CØØ88ØE5CØØ8AØE54
      ØØ88ØE5DØØC33A8D5FØØ
ND 9Ø DATA 7EF9888D5FØØ8A9D2BØØ8ØC3Ø68A871ØØØ2A85
      5FØØ3A855EØØ7C4Ø83FFØ174DB3A855EØØ7435C38A8
      D4CØØØ28D54ØØ8AD98A8767ØØ8Ø3E2BØØØØ75Ø63CØ1
      7DBB7CØ83CØØ7CB53CØ7
PF 1ØØ DATA 74B188852CØØ3CØ674Ø43CFA75ØAC6855FØØ2
       E5A5AEB5C9Ø8A9D4CØØ8A8767ØØC68767ØØØØ8AD98
       88767ØØ3B3E29ØØ75Ø3E9EFFE47C6854CØØ148Ø362
       BØØØ1FE854CØØ8A9D4CØØ8A
EK 11Ø DATA 9F67ØØ8Ø3E2BØØØØ75ØD8ØFBØ17C158ØFBØ77
       41ØEBØ89Ø8ØFBØØ7DØ8F6DBDØE3FF971BØØ8ØBD4CØ
       Ø627CCC83FFØØ74Ø98Ø362BØØØ14FE9A2FEAØ5CØØØ
       ØØ65DØØFBC38Ø3E2BØØØØ75
HO 12Ø DATA 5E8A9D4CØØ8ØC3ØA8ØBF67ØØØØ7523C68554Ø
       ØØAE838FF8A9D4CØØ8ØFB277D168ØC3148ØBF67ØØØ
       Ø75Ø8C68554ØØ14E81DFF8A9D4CØØ8ØC3Ø98ØBF67Ø
       ØØØ7DØ8C68554ØØØ9E8Ø7FF
DN 13Ø DATA 8A9D4CØØ8ØC3ØB8ØBF67ØØØØ7DØ8C68554ØØØ
       BE8F1FEC38A9D4CØØ8ØC3F68ØBF67ØØØØ7523C6855
       4ØØF6E8DAFE8A9D4CØØ8ØFB517C168ØC3EC8ØBF67Ø
       ØØØ75Ø8C68554ØØECE8BFFE
FL 14Ø DATA 8A9D4CØØ8ØC3F78ØBF67ØØØØ7EØ8C68554ØØF
       7E8A9FE8A9D4CØØ8ØC3F58ØBF67ØØØØ7EØ8C68554Ø
       ØF5E893FEC3C68534ØØØØB3ØØ8A87ØØØØ888554ØØE
       88ØFEFE8534ØØ8A9D34ØØ8Ø
OB 15Ø DATA FBØ87CE8C3C685440ØØ4C68534ØØØØEB189ØC
       68544ØØØ8C68534ØØØ4EBØB9ØC68544ØØØ8C68534Ø
       ØØØ8A9D34ØØ8A87Ø8ØØ88853CØØ888554ØØE83BFE8
       A854CØØØ28554ØØ8AD88ØBF
```

```
MI 16Ø DATA 67ØØØØ75ØA8A8554ØØØ2853CØØEBDEFE8534Ø
       Ø8A9D34ØØ3A9D44ØØ7CC8C3C68534ØØØØB3ØØ8A87Ø
       8ØØ888554ØØE8FFFDFE8534ØØ8A9D34ØØ8ØFBØ87CE
       8C3ØØØØ15ØCF8EDEBF4Ø813
KB 17Ø DATA ØBF7F5Ø9ØAØ1F6FF2EØ9Ø5Ø3Ø3Ø1ØØØ1Ø3Ø3Ø
       5Ø92EAAØ16DØ28DØ29AØ2A7Ø2EEØ2
```

Program 2. Chess (Main Program)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KM 1Ø CO=&H1CØØ:DEF SEG=&HFFFF:IF PEEK(14)=253 TH
      EN CO=&H17ØØ:I=1
AB 2Ø DA=CO+49:DEF SEG=CO:BLOAD"chess.bld",Ø:IF I
       THEN POKE 3,23:POKE 16,23
JD 3Ø DEF SEG=DA:GOSUB 69Ø
GF 4Ø M=4Ø:N=158:K=21
NF 5Ø POKE 43,1-BB:GOTO 18Ø
KD 6Ø IF C2 THEN 18Ø
NC 7Ø POKE 223,Ø:DEF SEG=CO:SOUND 99,Ø:CALL ML:DE
      F SEG=DA
LO 8Ø IF PEEK(95)<229 AND PEEK(95)>15Ø THEN I=Ø:G
      OTO 12Ø
BF 9Ø K1=PEEK(92):K=PEEK(93):SOUND 5ØØ,1:GOSUB 11
      9Ø:GOSUB 95Ø
QI 1ØØ IF PEEK(95)>99 OR PEEK(95)<28 THEN 18Ø
NF 11Ø I=1
OF 12Ø X=I+BB+PEEK(43):IF I=Ø THEN POKE 43,-(PEEK
       (43)=Ø)
BN 13Ø GOSUB 141Ø:PRINT"Checkmate!  ";
LO 14Ø IF X/2-INT(X/2) THEN PRINT"White wins.":GO
       TO 16Ø
IB 15Ø PRINT"Black wins."
FD 16Ø SOUND 999,9:FOR J=Ø TO 2ØØ:NEXT
HP 17Ø SOUND 26Ø,9:FOR J=Ø TO 2ØØ:NEXT
KG 18Ø F=Ø:M=M-8:N=N-3
NB 19Ø GOSUB 68Ø
NM 2ØØ C$=INKEY$:IF C$="" THEN 2ØØ
EP 21Ø IF LEN(C$)=1 THEN 27Ø
PO 22Ø C=ASC(RIGHT$(C$,1)):IF C=75 AND M>32 THEN
       GOSUB 68Ø:M=M-31:K=K-1:GOTO 19Ø
KB 23Ø IF C=77 AND M<249 THEN GOSUB 68Ø:M=M+31:K=
       K+1:GOTO 19Ø
DN 24Ø IF C=72 AND N>8 THEN GOSUB 68Ø:N=N-21:K=K+
       1Ø:GOTO 19Ø
DB 25Ø IF C=8Ø AND N<155 THEN GOSUB 68Ø:N=N+21:K=
       K-1Ø:GOTO 19Ø
BD 26Ø GOTO 2ØØ
QP 27Ø C=ASC(C$):GOSUB 14ØØ:IF C<>13 OR F=Ø THEN
       36Ø
```

```
GF 28Ø POKE 92,K1:POKE 93,K:J=PEEK(41):POKE 41,1:
       POKE 223,1
FF 29Ø DEF SEG=CO:CALL ML:DEF SEG=DA
DO 3ØØ POKE 41,J:IF PEEK(224)=Ø THEN 32Ø
CK 31Ø GOSUB 119Ø:GOSUB 95Ø:GOTO 6Ø
MG 32Ø X=PEEK(1Ø3+K1):IF (X=6 OR X=25Ø) AND ABS(K
       -K1)=2 THEN GOSUB 119Ø:GOSUB 95Ø:Y=K1:K1=2
       1-7Ø*(X>6)-7*(K>K1):K=K+(K>Y)-(Y>K):MM=MM-
       1:GOSUB 119Ø:PR(MV)=1:GOSUB 95Ø:GOTO 6Ø
DO 33Ø IF PEEK(1Ø3+K) THEN 35Ø
MI 34Ø IF (X=1 OR X=255) AND (ABS(K-K1)=9 OR ABS(
       K-K1)=11) THEN GOSUB 119Ø:GOSUB 95Ø:K=K+1Ø
       *(X=1)-1Ø*(X>1):MM=MM-1:GOSUB 119Ø:PR(MV)=
       1:GOSUB 95Ø:GOTO 6Ø
PE 35Ø SOUND 1ØØ,4:F=Ø:POKE 43,-(PEEK(43)=Ø):GOTO
        2ØØ
GA 36Ø IF F THEN 2ØØ
JO 37Ø IF C<>13 OR PEEK(1Ø3+K)=Ø THEN 41Ø
GJ 38Ø IF PEEK(43) AND PEEK(1Ø3+K)<7 THEN 4ØØ
DH 39Ø IF PEEK(43) OR PEEK(1Ø3+K)<7 THEN 41Ø
NO 4ØØ K1=K:F=1:SOUND 5ØØ,1:GOTO 2ØØ
AD 41Ø S=Ø
JL 42Ø IF D(S)=C THEN 45Ø
EN 43Ø S=S+1:IF S<28 THEN 42Ø
BB 44Ø GOTO 2ØØ
JA 45Ø IF S>22 THEN SOUND 5ØØ,1:LOCATE 1,22:PRINT
        C$:POKE 41,VAL(C$):GOTO 2ØØ
HA 46Ø IF S=13 THEN SOUND 5ØØ,1:GOSUB 68Ø:M=M+8:N
       =N+3:GOTO 7Ø
IF 47Ø IF S=14 THEN SOUND 5ØØ,1:FOR I=Ø TO 7Ø STE
       P 1Ø:FOR J=Ø TO 7:POKE 124+I+J,Ø:NEXT:NEXT
       :MX=Ø:MV=Ø:MM=Ø:BB=Ø:GOSUB 9ØØ:GOTO 4Ø
NN 48Ø IF S<>15 OR MV=Ø THEN 53Ø
LJ 49Ø SOUND 5ØØ,1:POKE 43,-(PEEK(43)=Ø):GOSUB 68
       Ø:GOSUB 12ØØ:MM=MM-1:GOSUB 143Ø
GD 5ØØ IF ABS(PC(MV)-128)=122 AND ABS(FR(MV)-T(MV
       ))=2 THEN GOSUB 12ØØ
FG 51Ø IF ABS(PC(MV)-128)=127 AND PC(MV+1)=Ø AND
       MV<MX THEN GOSUB 12ØØ
GM 52Ø GOTO 18Ø
HM 53Ø IF S<>16 OR MV>=MX THEN 58Ø
FH 54Ø SOUND 5ØØ,1:POKE 43,-(PEEK(43)=Ø):GOSUB 68
       Ø:GOSUB 121Ø:MM=MM+1:GOSUB 143Ø
LO 55Ø IF ABS(PC(MV)-128)=122 AND ABS(FR(MV)-T(MV
       ))=2 THEN GOSUB 121Ø
KC 56Ø IF ABS(PC(MV)-128)=127 AND PC(MV+1)=Ø AND
       MV<MX THEN GOSUB 121Ø
GG 57Ø GOTO 18Ø
NJ 58Ø IF S=17 THEN BB=Ø:GOTO 67Ø
```

```
PO 590 IF S=18 THEN 1280
HA 600 IF S=19 THEN 1220
FE 610 IF S=20 THEN 1340
KL 620 IF S=21 THEN BB=1:GOTO 670
MG 630 IF S=22 THEN SOUND 500,1:C2=1-C2
BI 640 IF S>12 THEN 200
HN 650 SOUND 500,1:IF S>6 THEN S=262-S
PI 660 POKE 103+K,S:GOSUB 950:M=M-8:N=N-3:GOTO 19
       0
EG 670 SOUND 500,1:MV=0:MM=0:FOR I=0 TO 77:POKE I
       +124,BD(I):NEXT:GOSUB 890:GOTO 40
QL 680 PUT (M,N),F,XOR:RETURN
NH 690 KEY OFF:SCREEN 1,0:COLOR 0,1:CLS
BG 700 POKE 41,1
AB 710 DEFINT P,N,B,R,Q,K,F
LD 720 DIM A(64),C(64),D(27),P(30),N(30),B(30),R(
       30),Q(30),K(30),F(82),FR(200),T(200),PC(20
       0),CA(200),PR(200),BD(77)
PP 730 FOR I=0 TO 27:READ D(I):NEXT
GJ 740 LINE (0,0)-(29,19),1,BF
NF 750 GET (0,0)-(29,19),A:CLS
HF 760 LINE (0,0)-(29,19),2,BF
AH 770 GET (0,0)-(29,19),C:CLS
QO 780 LOCATE 10,18:PRINT "CHESS"
ON 790 LOCATE 12,15:PRINT"John Krause"
MN 800 FOR I=103 TO 222:POKE I,7:NEXT
GL 810 FOR I=0 TO 77:READ BD(I):POKE I+124,BD(I):
       NEXT
PN 820 FOR K=0 TO 30:READ P(K):NEXT
MP 830 FOR K=0 TO 30:READ N(K):NEXT
KB 840 FOR K=0 TO 30:READ B(K):NEXT
CD 850 FOR K=0 TO 30:READ R(K):NEXT
BN 860 FOR K=0 TO 30:READ Q(K):NEXT
IP 870 FOR K=0 TO 30:READ K(K):NEXT
PL 880 FOR K=0 TO 82:READ F(K):NEXT:CLS
JC 890 IF BB THEN POKE 127,6:POKE 128,5:POKE 197,
       250:POKE 198,251
HM 900 LOCATE 1,5:PRINT"Move#        Level"PEEK(41)
       "  To move:":GOSUB 1430
EE 910 FOR I=0 TO 7:FOR J=0 TO 7
IM 920 H=70-10*I+J:GOSUB 960:NEXT:NEXT
CD 930 FOR I=1 TO 8:LOCATE 3*I-1+(I>4),2:PRINT 9-
       I:NEXT
DI 940 GOSUB 1400:RETURN
OO 950 H=K-21:I=INT(H/10):J=H-10*I:I=7-I
PA 960 M=31*J+40:N=21*I+11
FF 970 IF INT((I+J)/2)-(I+J)/2 THEN PUT (M-8,N-3)
       ,C,PSET: GOTO 990
MI 980 PUT (M-8,N-3),A,PSET
```

```
HC 99Ø L=PEEK(124+H):IF I=Ø AND L=1 THEN L=5:POKE
       124+H,L
JF 1ØØØ IF I=7 AND L=255 THEN L=251:POKE 124+H,L
PM 1Ø1Ø IF L>6 THEN L=L-256
OM 1Ø2Ø ON ABS(L) GOTO 1Ø4Ø,1Ø5Ø,1Ø6Ø,1Ø7Ø,1Ø8Ø,1
       Ø9Ø
IL 1Ø3Ø GOTO 11ØØ
NF 1Ø4Ø PUT (M,N),P,OR:GOTO 11ØØ
MI 1Ø5Ø PUT (M,N),N,OR:GOTO 11ØØ
QL 1Ø6Ø PUT (M,N),B,OR:GOTO 11ØØ
QO 1Ø7Ø PUT (M,N),R,OR:GOTO 11ØØ
PB 1Ø8Ø PUT (M,N),Q,OR:GOTO 11ØØ
IG 1Ø9Ø PUT (M,N),K,OR
PG 11ØØ IF BB THEN L=-L
KF 111Ø IF L>=Ø THEN RETURN
PF 112Ø ON -L GOTO 113Ø,114Ø,115Ø,116Ø,117Ø,118Ø
FD 113Ø PUT (M,N),P,XOR:RETURN
DG 114Ø PUT (M,N),N,XOR:RETURN
IJ 115Ø PUT (M,N),B,XOR:RETURN
IM 116Ø PUT (M,N),R,XOR:RETURN
HP 117Ø PUT (M,N),Q,XOR:RETURN
BC 118Ø PUT (M,N),K,XOR:RETURN
QN 119Ø K2=K:K=K1:MV=MV+1:PR(MV)=Ø:MM=MM+1:MX=MV:
       FR(MV)=K:PC(MV)=PEEK(1Ø3+K):POKE 1Ø3+K,Ø:
       GOSUB 95Ø:K=K2:T(MV)=K:CA(MV)=PEEK(1Ø3+K)
       :POKE 1Ø3+K,PC(MV):GOSUB 143Ø:RETURN
QN 12ØØ POKE 1Ø3+FR(MV),PC(MV):POKE 1Ø3+T(MV),CA(
       MV):K=T(MV):GOSUB 95Ø:K=FR(MV):GOSUB 95Ø:
       MV=MV-1:RETURN
FO 121Ø MV=MV+1:POKE 1Ø3+T(MV),PEEK(1Ø3+FR(MV)):P
       OKE 1Ø3+FR(MV),Ø:K=FR(MV):GOSUB 95Ø:K=T(M
       V):GOSUB 95Ø:RETURN
DM 122Ø SOUND 5ØØ,1:GOSUB 141Ø:INPUT"Save:",N$
KO 123Ø ON ERROR GOTO 142Ø
LA 124Ø OPEN N$+".chs" FOR OUTPUT AS #1
NC 125Ø FOR I=124 TO 2Ø1:PRINT #1,PEEK(I):NEXT
GM 126Ø PRINT #1,PEEK(41),PEEK(43),MV,MX,MM,BB,M,
       N,K,C2
BO 127Ø FOR I=1 TO MX:PRINT #1,T(I),FR(I),PC(I),C
       A(I),PR(I):NEXT:CLOSE #1:ON ERROR GOTO Ø:
       GOSUB 14ØØ:GOTO 2ØØ
BC 128Ø SOUND 5ØØ,1:GOSUB 141Ø:INPUT"Load:",N$
LA 129Ø ON ERROR GOTO 142Ø
GN 13ØØ OPEN N$+".chs" FOR INPUT AS #1
FJ 131Ø FOR I=124 TO 2Ø1:INPUT #1,J:POKE I,J:NEXT
EM 132Ø INPUT #1,X,J,MV,MX,MM,BB,M1,N1,K1,C2:POKE
        41,X:POKE 43,J
MI 133Ø FOR I=1 TO MX:INPUT #1,T(I),FR(I),PC(I),C
       A(I),PR(I):NEXT:CLOSE #1:ON ERROR GOTO Ø:
       GOSUB 9ØØ:M=M1:N=N1:K=K1:GOTO 19Ø
```

```
PC 134Ø SOUND 5ØØ,1:X=Ø:FOR I=1 TO MX:IF PR(I) TH
        EN 137Ø
BD 135Ø X=X+1:IF X/2-INT(X/2) THEN LPRINT(X+1)/2"
         ";:GOSUB 138Ø:GOTO 137Ø
JH 136Ø LPRINT"    ";:GOSUB 138Ø:LPRINT
HH 137Ø NEXT:LPRINT:GOTO 2ØØ
HE 138Ø J=INT(FR(I)/1Ø):LPRINT CHR$(64+FR(I)-1Ø*J
        );MID$(STR$(J-1),2,1)"-";
MD 139Ø J=INT(T(I)/1Ø):LPRINT CHR$(64+T(I)-1Ø*J);
        MID$(STR$(J-1),2,1);:RETURN
LC 14ØØ LOCATE 23,6:PRINT"A   B   C   D   E   F
         G   H":RETURN
LE 141Ø LOCATE 23,6:PRINT"
             ":LOCATE 23,9:RETURN
LD 142Ø GOSUB 141Ø:PRINT"Error #"ERR:RESUME 2ØØ
LF 143Ø LOCATE 1,1Ø:PRINT INT(MM/2+1)" ":LOCATE 1
        ,35:IF INT(MM/2)=MM/2 THEN PRINT CHR$(87)
        :RETURN
FE 144Ø PRINT CHR$(66):RETURN
NB 145Ø DATA 32,8Ø,78,66,82,81,75,16,14,2,18,17,1
        1,1Ø9,99,98,1Ø2,11Ø,1Ø8,115,112,1Ø5,116,4
        9,5Ø,51,52,53
CL 146Ø DATA 4,2,3,5,6,3,2,4,7
FO 147Ø DATA 7,1,1,1,1,1,1,1,1,7
LB 148Ø DATA 7,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,7
LE 149Ø DATA 7,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,7
KL 15ØØ DATA 7,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,7
KO 151Ø DATA 7,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,7
BJ 152Ø DATA 7,255,255,255,255,255,255,255,255,7
HO 153Ø DATA 7,252,254,253,251,25Ø,253,254,252
BO 154Ø DATA 28,14,Ø,Ø,Ø,Ø,384Ø,Ø
OE 155Ø DATA 16128,192,16128,192,384Ø,Ø,16128,192
AF 156Ø DATA 384Ø,Ø,384Ø,Ø,16128,192,-256,24Ø
DG 157Ø DATA -256,24Ø,Ø,Ø,Ø,Ø,128
PK 158Ø DATA 28,14,3,Ø,-16381,Ø,-1Ø21,Ø
LC 159Ø DATA-241,192,-244,24Ø,-241,24Ø,-241,252
PL 16ØØ DATA -193,252,-12481,255,3852,255,16128,2
        55
KF 161Ø DATA -256,255,-253,255,-253,255,-253
PB 162Ø DATA 28,14,-4Ø96,24Ø,-4Ø96,24Ø,-1Ø21,252
CF 163Ø DATA -253,6Ø,-253,2Ø4,-253,2Ø4,-253,2Ø4
KK 164Ø DATA -256,24Ø,-16384,48,-256,24Ø,-16384,4
        8
LL 165Ø DATA -193,-16129,-3841,-3841,192,12288,-2
        53
OP 166Ø DATA 28,14,16143,2Ø7,16143,2Ø7,-241,255
IJ 167Ø DATA 3,12,-253,252,-253,252,-253,252
NO 168Ø DATA -253,252,-253,252,-253,252,3,12
EG 169Ø DATA -241,255,-193,-16129,-193,-16129,-19
        3
```

```
CE 1700 DATA 28,14,-16384,192,-16384,192,-16384,1
        92
FA 1710 DATA -16192,-16192,-3133,-16144,-3277,243
        ,-3277,243
ED 1720 DATA -193,255,12,12,-241,252,-3313,252
KI 1730 DATA -241,252,12,12,-241,252,0
CD 1740 DATA 28,14,-256,192,-13312,192,-3268,207
KP 1750 DATA -13057,-16129,-1,-16129,-16129,-1612
        9,-3265,255
FP 1760 DATA -193,255,12,12,-241,252,-3313,252
KE 1770 DATA -241,252,12,12,-241,252,0
CJ 1780 DATA 60,20,-1,-1,-1,-3841,-1,-1
GB 1790 DATA -1,-3841,252,0,0,-4093,252,0
DF 1800 DATA 0,-4093,252,0,0,-4093,252,0
EI 1810 DATA 0,-4093,252,0,0,-4093,252,0
EL 1820 DATA 0,-4093,252,0,0,-4093,252,0
EO 1830 DATA 0,-4093,252,0,0,-4093,252,0
EB 1840 DATA 0,-4093,252,0,0,-4093,252,0
EE 1850 DATA 0,-4093,252,0,0,-4093,252,0
EH 1860 DATA 0,-4093,252,0,0,-4093,252,0
II 1870 DATA 0,-4093,-1,-1,-1,-3841,-1,-1
IF 1880 DATA -1,-3841,0
```

CHAPTER 3

Applications

Home Financial Calculator

Patrick Parrish

Many home budget programs have been published in magazines and books, but rarely has there been a program integrating as wide a variety of loan and investment calculations as "Home Financial Calculator." It is versatile, easy to use, and flexible. Rapid recalculation features make it an ideal tool for "what if?" projections. A calculator mode with memory lets you solve problems not directly supported by the program, and you can pass values generated by one calculation to another.

Investment and loan calculations are readily computerized. In fact, many programs have been written which perform these tasks individually. "Home Financial Calculator" goes a step further by integrating several common financial calculations in a menu-driven package. It also features a calculator mode, or scratch pad area, where program variables can be manipulated using common mathematical operations.

When you run the program, a main menu offers you a choice of Investment or Loan calculations. Type I or L to reach the appropriate submenu.

Common Variables

Before looking at any calculations, let's consider some basics of the program. Home Financial Calculator uses some parameters or variables repeatedly in the calculations. These variables are *Total* (also referred to as Future Value, Total Owed, and so on, depending on the calculation), *Present Value* (principal), *Interest Rate, Years, Months, Number of Periods* (of either compounding deposits, withdrawals, or payments, depending on the application), *Deposits,* and *Withdrawals.* When in the calculator mode (explained below), you'll reference these eight variables with the single letters T, P, I, Y, M, N, D, and W.

As you work with Home Financial Calculator, the values of the eight variables are preserved until you change them. Whenever the program asks you for an input (for example, Interest), the current value of that variable is displayed (zero if

no value has been entered yet). If you want to keep the current value, just press Enter. Otherwise, enter the new value and press Enter.

With this feature, Home Financial Calculator makes it easy for you to generate "what if?" projections. Simply run the same calculation repeatedly, each time changing a previously entered value. Press Enter to keep a value, and change only one or two values to see the effect on the final result.

You can also store the current value into the calculator mode's Memory Register or recall a value from the Memory Register. To see how all this works, let's take a look at some calculations possible with Home Financial Calculator.

Investment Calculations

Here is the Investment submenu that appears when you type I from the main menu:

1) Future Value with Periodic Interest
2) Future Value with Interest Compounded Continuously
3) Future Value with Regular Deposits
4) Future Value vith Cash Flows
5) Withdrawal of Funds
6) Net Present Value
7) Calculator Mode
8) Enter to Main Menu.

Determine which option you want and press the appropriate key.

Each option displays screen prompts which ask you to input several values. These values are stored in the eight variables mentioned above: *T* for Total (Future Value), *P* for Present Value (principal), *I* for Interest Rate, *Y* for Years, *M* for Months, *N* for Number of Periods, *D* for Deposits, and *W* for Withdrawals. Of course, not all calculations require you to enter all these values, while others may ask for additional information.

Most calculations can be solved for any *one* of the variables. To solve for a variable, enter an uppercase X at the corresponding input prompt. For example, you could enter values for everything except the Interest Rate, typing X at the Interest Rate prompt. Home Financial Calculator then solves for the Interest Rate.

Remember, however, that the program can solve for only

one variable during each calculation. If you enter an X at more than one prompt, the program does not have enough information to calculate an answer. Keep this in mind, because the program does not check for potential conflicts.

Future Value with Periodic Interest. Home Financial Calculator's options are fairly self-explanatory when you run the program, but let's try an example. We'll calculate the future value of an investment drawing periodic interest. This kind of investment could be a savings account, interest-bearing checking account, bonds, or a money market account. Choose this option by entering 1 at the Investment submenu.

After the screen clears, the program asks for the first input—Future Value, which appears with an asterisk (*). Below this is a zero (the current value of this variable in memory; all variables start out with a value of zero). Following this is an input prompt.

The asterisk preceding Future Value means that this is one of the variables you can solve for. (A variable *not* preceded by an asterisk means that variable *cannot* be solved for in that particular calculation, so X would be an illegal response.) If you'd like to calculate the Future Value, enter an X here, and answer all the other prompts with the appropriate values.

Let's calculate the future value of a $1,000 investment drawing 8 percent interest for two years and three months, with four compounding periods each year. Enter an X for *Future Value,* since we'll be solving for this total. Answer *Present Value* with 1000 (the principal you're investing); *Annual Int Rate (%)* with 8 (enter the percentage, not a fraction); *For # Of Years* with 2; *For # Of Months* with 3; and *# Of Periods (Compounding)* with 4. After you enter the last value, Home Financial Calculator figures the *Total Future Value* and displays the answer—$1195.09.

Now suppose you wish to know the future value of the same $1,000 investment if you make 9 percent interest. Choose option 1 on the Investment submenu again and rerun the calculation. Notice how Home Financial Calculator automatically prints the current value of each variable at each prompt. The *Future Value* prompt shows a current value of 1195.09 from the previous calculation. Type X at this prompt and 9 for Interest Rate, and press Enter at all other prompts to preserve their values. The result should be $1221.71.

The versatility of Home Financial Calculator becomes apparent when you realize how many different ways you can run this calculation. Using this same menu option, you can calculate the initial investment (or present value) necessary to accrue a certain future value with periodic interest; the interest rate necessary to accrue a future value from a present value; or the time (in years and months) it would take to accumulate a future amount from an initial investment with periodic interest payments. Just enter an X for the unknown value you're seeking, and fill in all the other prompts.

Future Value with Interest Compounded Continuously. Option 2, a variation of option 1, handles investments paying a continuous interest rate. Like option 1, option 2 can handle a number of calculations—just place an X in the slot you'd like to solve for.

Here, after entering all other parameters, you can calculate the future value of an investment, the initial investment required to reach a certain future value, the interest required to reach a desired future value, or the time required to reach a certain future value at a specified interest rate.

Notice that any variables used in option 1 will be displayed with their current values when running option 2. As mentioned above, the eight major variables in Home Financial Calculator retain their values throughout the program until you change them. This feature is convenient when going from one option to another on the Investment or Loan submenus.

In addition, the values are preserved for use in the calculator mode. For instance, you could compare the effect of continuously compounded interest with periodic interest (option 1) without having to retype the input.

Future Value with Regular Deposits. If you're interested in setting up an annuity, you'd choose option 3 on the Investment submenu. You can determine the future value of an account (such as a savings account, Individual Retirement Account, or college or vacation fund) with regular deposits where interest is compounded with each deposit.

Option 3 can also tell you the amount of each deposit necessary to accrue a future value, the interest rate needed to provide some future value with regular deposits, or the time it would take to amass a future value with regular deposits.

Future Value with Cash Flows. Option 4 does a single calculation—it always solves for *Future Value,* so don't enter

an X anywhere. It calculates the future value of an investment with yearly cash flows (either positive or negative). The *Annual Interest Rate* you input here is the growth rate on the money you've invested.

As an example, suppose you wish to determine the value of a vacation fund collected over four years. You're asked for the number of years, then for the deposit or withdrawal each year. You deposit $500 in the fund the first year and $200 the second. The third year you are forced to withdraw $300 (entered as −300), and the fourth year, you put in $400. The fund has a growth rate of 12 percent. Its value after four years will be $1,017.34.

A future value determination can also tell you whether an investment is worthwhile. If the future value of all cash flows is positive or zero, the investment is profitable. A negative future value, on the other hand, represents a losing investment.

Withdrawal of Funds. If you intend to open an account from which you can regularly withdraw funds, choose option 5. With this option, you can determine the initial deposit required in the account to cover your withdrawals, the amount you can withdraw regularly from this account, the rate of interest you must make on funds in the account, or the period of time over which you can make withdrawals.

Net Present Value. Option 6 lets you determine the feasibility of a prospective investment by calculating its net present value. Net present value is the current value of all future yearly cash flows to an investment along with any initial cash requirement. The interest rate you input here is the rate of return you require on your investment. A positive net present value indicates a profitable investment, while a negative result signifies a losing investment.

As an example, suppose you have the opportunity to make a $2,000 investment which would return $1,500 the first year, cost you $750 the second year, and return $1,900 the third year. You hope to make 13 percent on your money. With option 6, you determine a net present value of $56.87, representing a profitable investment.

Calculator Mode. Option 7 puts you in the calculator mode (also available from the Loan submenu). Calculator mode works very much like a handheld calculator with a single memory. You can type in a value or recall one from a variable by entering its symbol—T(otal), P(resent Value), I(nterest

Rate), Y(ears), M(onths), N(umber of Periods), D(eposits), and W(ithdrawals). You can perform simple math on values stored in the Memory Register using reverse Polish notation. And you can use the results in future calculations.

When you enter calculator mode, the calculator command line appears on the screen:

V S H R M+ M− M* M/ MR MC
MEM=0

Here are the commands:

- **V** View the values of the eight primary variables
- **S** Store Memory Register into a variable
- **H** Help—prints the command line
- **R** Return to main menu, exit calculator mode
- **M+** Add the last input to the Memory Register
- **M−** Subtract the last input from the value in the Memory Register and store the result in the Register
- **M*** Multiply the last input times the value in the Memory Register and store the result in the Register
- **M/** Divide the last input into the value in the Memory Register and store the result in the Register
- **MR** Memory Recall
- **MC** Memory Clear to zero

If you've run through a sample investment calculation, you now have some variables in memory. Enter V in the calculator mode to see them. The screen displays the eight values currently in memory for the eight variables.

To work with one of these variables, enter one of their letters (T, P, I, Y, M, N, D, or W), and press Enter. Then type M+ to add it to the Memory Register (all variables must be stored in the Register before you can perform any operations on them). Suppose you put the current value for T into the Register and now wish to add $229 to this value. Enter 229, press Enter, then type M+, and press Enter. The addition is performed and the result displayed. To store this value back into the T variable, enter S for Store. A prompt appears, requesting the variable in which you intend to store the value. Type T to store the value into the variable T.

You can also use the Memory Register to hold a value not represented by any of the eight variables. To do this, determine a value using the calculator mode and store it into the Memory Register with M+. Then, when you're running a calculation elsewhere in the program, you can substitute this

value for any of the eight primary variables by typing MR (Memory Recall) at the appropriate prompt. MR can be used both in the calculator mode and at any prompt where the previous value is displayed.

Finally, option 8 on the Investment submenu returns you to the main menu. Once there, you can perform some loan calculations by typing L.

Loan Calculations

Here is the Loan calculations submenu:

1) Regular Loan Payments
2) Remaining Loan Liability
3) Final Loan Payment
4) Single Payment Loan
5) Loan Amortization Schedule
6) Calculator Mode
7) Return to Main Menu

Regular Loan Payments. Option 1 handles a number of calculations for equal payment loans. You can figure the principal of a loan, the amount of each regular payment necessary to repay a loan, the annual interest rate on a loan with regular payments, or the term of the loan.

Remaining Loan Liability. With option 2, you can determine the remaining balance on a loan with regular payments after a number of payments have been made. Enter the principal on the loan, the amount of each payment, the annual interest rate, the number of payments yearly, and the last payment number.

Final Loan Payment. Option 3 calculates the amount of the final payment on a loan. In many cases, the last payment of a loan will vary from the amount of the regular payment. This option handles situations where the final payment is greater than ("balloon payments") or less than the regular payment.

Single Payment Loan. Option 4 calculates the amount owed on a loan that is paid off with a single payment. You must input the principal on the loan, its annual interest rate, its term in years and months, and the number of times a year the interest on the principal is compounded.

Loan Amortization Schedule. Option 5 displays a loan amortization schedule. Enter the principal on the loan, the amount of each payment, the annual interest rate, the term of

the loan, and the number of payments yearly. Then enter the period of the year in which the loan began (for instance, 10 for October) and the range in years of the amortization schedule you'd like to examine.

Because of the complexity of these calculations, there may be a delay before the output appears on the screen, especially if you have chosen to look at the latter years in a long-term loan repayment schedule (such as a home mortgage). When the amortization table appears, it displays the payment number, the beginning balance for the period, the amount paid toward the loan principal, the amount paid in interest, and the ending balance. To keep the information from scrolling off the screen, the program shows only a few payment periods at a time. Press Enter to view another screenful. When the end of a year is reached, the program gives the total amounts paid on the principal and in interest for the year. In addition, when the last period of the loan is reached, the program displays the final payment for the loan.

The last two options on the Loan submenu are the same as those on the Investment submenu.

Modifying the Program

Home Financial Calculator is written in a modular format for easy modification. For many routines, it uses common input labels (lines 4710–5080) and some output labels (lines 5090–5170). If you want to add an investment or loan calculation routine, choose the labels from these lines that fit your application.

Also, you may wish to add a printer option to the loan amortization schedule. Examine lines 3230–3940. Here, variable D5 (defined in line 150) determines the number of loan payments considered on each screen. Variables S1, S2, S3, and S4 (defined in lines 160–190) format the output horizontally on the screen.

Home Financial Calculator

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
BD 9Ø WIDTH 4Ø:KEY OFF:DEF SEG=Ø:POKE 1Ø47,PEEK(1
      Ø47) OR 64
LA 1ØØ DIM V(8)
CC 11Ø V$="TPIYMNDW"
HF 12Ø C$="VSHR"
IM 13Ø C1$="M+M-M*M/MRMC"
```

```
AD 140 Q$=""
EL 150 D5=13
MN 160 S1=4
IB 170 S2=14
IP 180 S3=22
IN 190 S4=30
GG 200 GOSUB 5450
MF 210 PRINT "INVESTMENTS OR LOANS"
FL 220 PRINT "(I/L) ";
HA 230 INPUT A$
BN 240 IF A$="I" THEN 270
KP 250 IF A$="L" THEN 2170
DE 260 GOTO 230
HE 270 GOSUB 5450
EJ 280 PRINT "INVESTMENTS:"
ND 290 PRINT
FA 300 PRINT "1) FUTURE VALUE WITH PERIODIC INTER
       EST"
QI 310 PRINT "2) FUTURE VALUE WITH INTEREST COMPO
       UNDED CONTINUOUSLY"
OM 320 PRINT "3) FUTURE VALUE WITH REGULAR DEPOSI
       TS"
PJ 330 PRINT "4) FUTURE VALUE WITH CASH FLOWS"
DN 340 PRINT "5) WITHDRAWAL OF FUNDS"
NK 350 PRINT "6) NET PRESENT VALUE"
IF 360 PRINT "7) CALCULATOR MODE"
CK 370 PRINT "8) RETURN TO MAIN MENU"
NC 380 PRINT
NG 390 PRINT "CHOICE ";
HM 400 INPUT A$
IH 410 A=VAL(A$)
GL 420 IF A<1 THEN 400
LF 430 IF A>8 THEN 400
LH 440 ON A GOTO 470,730,970,1360,1550,1940,450,2
       00
HK 450 GOSUB 4180
BF 460 GOTO 200
HG 470 GOSUB 5450
GH 480 PRINT "FUTURE VALUE WITH PERIODIC INTEREST
       "
NF 490 PRINT
FO 500 GOSUB 4710
IE 510 GOSUB 4750
DK 520 PRINT "*";
IH 530 GOSUB 4840
DO 540 PRINT "*";
MP 550 GOSUB 4880
FM 560 IF E=4 THEN 580
IB 570 GOSUB 4920
ME 580 GOSUB 4970
```

```
HE 590 IF E<>1 THEN 620
PH 600 V(1)=INT(V(2)*(1+V(3)/V(6))^(V(6)*Y)*100+.
       5)/100
HA 610 GOSUB 5090
LL 620 IF E<>2 THEN 650
AG 630 V(2)=INT(V(1)/((1+V(3)/V(6))^(V(6)*Y))*100
       +.5)/100
CH 640 GOSUB 5120
PF 650 IF E<>3 THEN 680
PL 660 V(3)=INT((V(6)*(V(1)/V(2))^(1/(V(6)*Y))-V(
       6))*10000+.5)/10000
FE 670 GOSUB 5150
JD 680 IF E<>4 THEN 710
DL 690 V(4)=LOG(V(1)/V(2))/(V(6)*LOG(1+V(3)/V(6))
       )
HO 700 GOSUB 5180
EH 710 GOSUB 5330
GN 720 GOTO 270
HB 730 GOSUB 5450
JL 740 PRINT "FUTURE VALUE WITH INTEREST COMPOUND
       ED CONTINUOUSLY"
NA 750 PRINT
GM 760 GOSUB 4710
JC 770 GOSUB 4750
EI 780 PRINT "*";
JF 790 GOSUB 4840
DJ 800 PRINT "*";
MK 810 GOSUB 4880
DC 820 IF E=4 THEN 840
IM 830 GOSUB 4920
PA 840 IF E<>1 THEN 870
JD 850 V(1)=INT(V(2)*EXP(V(3)*Y)*100+.5)/100
IM 860 GOSUB 5090
JO 870 IF E<>2 THEN 900
PK 880 V(2)=INT(V(1)/EXP(V(3)*Y)*100+.5)/100
DD 890 GOSUB 5120
MF 900 IF E<>3 THEN 930
BE 910 V(3)=INT(LOG(V(1)/V(2))/Y*10000+.5)/10000
FN 920 GOSUB 5150
JM 930 IF E<>4 THEN 710
NP 940 V(4)=INT(LOG(V(1)/V(2))/V(3)*100+.5)/100
IK 950 GOSUB 5180
FH 960 GOTO 710
IL 970 GOSUB 5450
KM 980 PRINT "FUTURE VALUE WITH REGULAR DEPOSITS"
OK 990 PRINT
JG 1000 GOSUB 4710
GC 1010 PRINT "*REGULAR DEPOSIT $"
HE 1020 C=6
OF 1030 GOSUB 3950
```

```
GE 1Ø4Ø PRINT "*";
OM 1Ø5Ø GOSUB 484Ø
HK 1Ø6Ø PRINT "*";
CK 1Ø7Ø GOSUB 488Ø
NI 1Ø8Ø IF E=4 THEN 11ØØ
OJ 1Ø9Ø GOSUB 492Ø
AG 11ØØ GOSUB 497Ø
PK 111Ø IF E<>1 THEN 114Ø
GC 112Ø V(1)=INT(V(7)*V(6)*((1+V(3)/V(6))^(V(6)*Y
        )-1)/V(3)*1ØØ+.5)/1ØØ
MC 113Ø GOSUB 5Ø9Ø
HD 114Ø IF E<>3 THEN 128Ø
KE 115Ø V(3)=.99
HG 116Ø I=Ø
GK 117Ø T=INT(V(7)*(((1+V(3)/V(6))^(V(6)*Y)-1)/(V
        (3)/V(6)))*1ØØ+.5)/1ØØ
LH 118Ø TE=ABS(V(3)-I)/2
OI 119Ø I=V(3)
LK 12ØØ IF ABS(T-V(1))<.ØØ5 THEN 126Ø
FK 121Ø IF T<V(1) THEN 124Ø
PB 122Ø V(3)=V(3)-TE
OK 123Ø GOTO 117Ø
OJ 124Ø V(3)=V(3)+TE
OA 125Ø GOTO 117Ø
GB 126Ø V(3)=INT(V(3)*1ØØØØ+.5)/1ØØØØ
KF 127Ø GOSUB 515Ø
BO 128Ø IF E<>4 THEN 131Ø
GJ 129Ø V(4)=LOG(V(3)*V(1)/(V(6)*V(7))+1)/(V(6)*L
        OG(1+V(3)/V(6)))
MM 13ØØ GOSUB 518Ø
DE 131Ø IF E<>7 THEN 71Ø
IM 132Ø V(7)=INT(V(1)*(V(3)/V(6))/((1+V(3)/V(6))^
        (V(6)*Y)-1)*1ØØ+.5)/1ØØ
GM 133Ø PRINT
JH 134Ø PRINT "REGULAR DEPOSITS REQUIRED:$";V(7)
DJ 135Ø GOTO 71Ø
NL 136Ø GOSUB 545Ø
KG 137Ø PRINT "FUTURE VALUE WITH CASH FLOWS"
HL 138Ø PRINT
PO 139Ø GOSUB 484Ø
BN 14ØØ GOSUB 488Ø
NF 141Ø PRINT "CASH FLOW (+/-)"
GL 142Ø PRINT
CJ 143Ø V(1)=Ø
EG 144Ø FOR I=1 TO V(4)
EP 145Ø PRINT "CASH FLOW - YEAR #";I
II 146Ø INPUT A$
JG 147Ø A=VAL(A$)
MN 148Ø V(1)=V(1)+A*(1+V(3))^(V(4)-I)
HC 149Ø NEXT I
```

```
NF 1500 V(1)=INT(V(1)*100+.5)/100
ME 1510 GOSUB 5090
PB 1520 TE=V(1)
NI 1530 GOSUB 5270
DK 1540 GOTO 710
NM 1550 GOSUB 5450
MB 1560 PRINT "WITHDRAWAL OF FUNDS"
HM 1570 PRINT
PA 1580 GOSUB 4750
HL 1590 PRINT "*REGULAR WITHDRAWAL $"
IC 1600 C=7
PL 1610 GOSUB 3950
HK 1620 PRINT "*";
OC 1630 GOSUB 4840
HA 1640 PRINT "*";
CA 1650 GOSUB 4880
NN 1660 IF E=4 THEN 1680
OP 1670 GOSUB 4920
DI 1680 GOSUB 4970
GG 1690 IF E<>2 THEN 1720
HF 1700 V(2)=INT(V(8)*V(6)/V(3)*(1-(1+V(3)/V(6))^
        (-V(6)*Y))*100+.5)/100
HD 1710 GOSUB 5120
MH 1720 IF E<>3 THEN 1860
LK 1730 V(3)=.99
IM 1740 I=0
HL 1750 R=INT(V(2)*V(3)/V(6)*(1/((1+V(3)/V(6))^(V
        (6)*Y)-1)+1)*100+.5)/100
MN 1760 TE=ABS(V(3)-I)/2
OO 1770 I=V(3)
KH 1780 IF ABS(R-V(8))<.005 THEN 1840
BL 1790 IF R<V(8) THEN 1820
PH 1800 V(3)=V(3)-TE
BO 1810 GOTO 1750
OP 1820 V(3)=V(3)+TE
BE 1830 GOTO 1750
GH 1840 V(3)=INT(V(3)*10000+.5)/10000
LL 1850 GOSUB 5150
CA 1860 IF E<>4 THEN 1890
CO 1870 V(4)=LOG(V(6)*V(8)/(V(6)*V(8)-V(3)*V(2)))
        /(V(6)*LOG(1+V(3)/V(6)))
OO 1880 GOSUB 5180
GC 1890 IF E<>8 THEN 710
KK 1900 V(8)=INT(V(2)*V(3)/V(6)*(1/((1+V(3)/V(6))
        ^(V(6)*Y)-1)+1)*100+.5)/100
GC 1910 PRINT
OA 1920 PRINT "REGULAR WITHDRAWALS:$";V(8)
DP 1930 GOTO 710
NB 1940 GOSUB 5450
NP 1950 PRINT "NET PRESENT VALUE:$"
```

```
HB 1960 PRINT
CI 1970 PRINT "INITIAL INVESTMENT"
HA 1980 C=1
BJ 1990 GOSUB 3950
NO 2000 GOSUB 4840
BJ 2010 GOSUB 4880
NB 2020 PRINT "CASH FLOW (+/-)"
FH 2030 PRINT
MK 2040 NV=-V(2)
EC 2050 FOR I=1 TO V(4)
MB 2060 PRINT "CASH FLOW - YEAR # ";I
HE 2070 INPUT A$
JC 2080 A=VAL(A$)
II 2090 NV=NV+A/((V(3)+1)^I)
FC 2100 NEXT I
IP 2110 NV=INT(NV*100+.5)/100
FG 2120 PRINT
HD 2130 PRINT "NET PRESENT VALUE:$";NV
HN 2140 TE=NV
MH 2150 GOSUB 5270
DJ 2160 GOTO 710
NL 2170 GOSUB 5450
OK 2180 PRINT "LOANS:"
HL 2190 PRINT
FL 2200 PRINT "1) REGULAR LOAN PAYMENTS"
II 2210 PRINT "2) REMAINING LOAN LIABILITY"
HO 2220 PRINT "3) FINAL LOAN PAYMENT"
DN 2230 PRINT "4) SINGLE PAYMENT LOAN"
PL 2240 PRINT "5) LOAN AMORTIZATION SCHEDULE"
OO 2250 PRINT "6) CALCULATOR MODE"
KD 2260 PRINT "7) RETURN TO MAIN MENU"
GH 2270 PRINT
LG 2280 PRINT "CHOICE ";
IO 2290 INPUT A$
IA 2300 A=VAL(A$)
EE 2310 IF A<1 THEN 2290
KN 2320 IF A>7 THEN 2290
BK 2330 ON A GOTO 2360,2780,2960,3120,3230,2340,2
        00
MN 2340 GOSUB 4180
OH 2350 GOTO 200
NM 2360 GOSUB 5450
KK 2370 PRINT "REGULAR LOAN PAYMENTS"
HM 2380 PRINT
IK 2390 PRINT "*";
BP 2400 GOSUB 4790
GE 2410 PRINT "*";
FG 2420 GOSUB 5010
HK 2430 PRINT "*";
OC 2440 GOSUB 4840
```

```
HA 2450 PRINT "*";
CA 2460 GOSUB 4880
NN 2470 IF E=4 THEN 2490
OP 2480 GOSUB 4920
DI 2490 GOSUB 4970
HE 2500 IF E<>2 THEN 2550
GE 2510 V(2)=INT(V(7)*V(6)/V(3)*(1-(1+V(3)/V(6))^
        (-V(6)*Y))*100+.5)/100
GO 2520 PRINT
DB 2530 PRINT "AMT OF PRINCIPAL:$";V(2)
DK 2540 GOTO 2760
PH 2550 IF E<>3 THEN 2690
LA 2560 V(3)=.99
IC 2570 I=0
NP 2580 P=INT(V(7)*V(6)/V(3)*(1-((1+V(3)/V(6))^(-
        V(6)*Y)))*100+.5)/100
MD 2590 TE=ABS(V(3)-I)/2
NI 2600 I=V(3)
FD 2610 IF ABS(P-V(2))<.005 THEN 2670
MD 2620 IF P<V(2) THEN 2650
OP 2630 V(3)=V(3)+TE
DO 2640 GOTO 2580
AD 2650 V(3)=V(3)-TE
DE 2660 GOTO 2580
HN 2670 V(3)=INT(V(3)*10000+.5)/10000
LB 2680 GOSUB 5150
JC 2690 IF E<>4 THEN 2720
KF 2700 V(4)=-LOG(1-V(3)*V(2)/(V(6)*V(7)))/(V(6)*
        LOG(V(3)/V(6)+1))
NI 2710 GOSUB 5180
PH 2720 IF E<>7 THEN 2760
OM 2730 V(7)=INT(V(3)*V(2)/(V(6)*(1-(V(3)/V(6)+1)
        ^(-V(6)*Y)))*100+.5)/100
HI 2740 PRINT
BG 2750 PRINT "REQ PAYMENT:$";V(7)
LL 2760 GOSUB 5330
QM 2770 GOTO 2170
OK 2780 GOSUB 5450
JH 2790 PRINT "REMAINING LOAN LIABILITY"
GO 2800 PRINT
CK 2810 GOSUB 4790
GO 2820 GOSUB 5010
OH 2830 GOSUB 4840
CB 2840 GOSUB 4970
CD 2850 PRINT "LAST PAYMENT # WAS:"
KB 2860 C=8
AC 2870 GOSUB 3950
ME 2880 FOR J=1 TO V(9)
GO 2890 I=INT(P*V(3)/V(6)*100+.5)/100
GE 2900 P=P+I-V(7)
```

```
HA 2910 NEXT J
QM 2920 V(10)=INT(P*100+.5)/100
HJ 2930 PRINT
LD 2940 PRINT "LIABILITY AFTER ";V(9);" PAYMENTS:
        $";V(10)
DF 2950 GOTO 2760
OI 2960 GOSUB 5450
EN 2970 PRINT "LAST LOAN PAYMENT"
II 2980 PRINT
DE 2990 GOSUB 4790
FJ 3000 GOSUB 5010
NC 3010 GOSUB 4840
IH 3020 GOSUB 5050
BP 3030 GOSUB 4970
JI 3040 FOR J=1 TO V(6)*Y
ED 3050 I=INT(P*V(3)/V(6)*100+.5)/100
GF 3060 P=P+I-V(7)
HB 3070 NEXT J
QK 3080 V(11)=INT(P*100+.5)/100+V(7)
HK 3090 PRINT
EK 3100 PRINT "LAST PAYMENT:$";V(11)
CK 3110 GOTO 2760
MN 3120 GOSUB 5450
CG 3130 PRINT "SINGLE PAYMENT LOAN"
GN 3140 PRINT
CJ 3150 GOSUB 4790
OD 3160 GOSUB 4840
KI 3170 GOSUB 5050
CA 3180 GOSUB 4970
HP 3190 V(1)=INT(V(2)*(1+V(3)/V(6))^(Y*V(6))*100+
        .5)/100
FD 3200 PRINT
LJ 3210 PRINT "TOTAL OWED:$";V(1)
CP 3220 GOTO 2760
DN 3230 C5=0
HC 3240 N5=0
GF 3250 F=0
HI 3260 P1=0
EB 3270 I1=0
NB 3280 GOSUB 5450
MO 3290 PRINT "LOAN AMORTIZATION SCHEDULE"
FF 3300 PRINT
BB 3310 GOSUB 4790
FF 3320 GOSUB 5010
OO 3330 GOSUB 4840
JD 3340 GOSUB 5050
CF 3350 PRINT "# OF PAYMENTS YEARLY"
PG 3360 GOSUB 3950
AE 3370 PRINT "ENTER THE PERIOD OF THE YEAR IN WH
        ICH THE LOAN BEGAN"
```

```
EG 3380 INPUT N
AB 3390 NE=N
NP 3400 NP=(V(4)*12+V(5))/(12/V(6))
JN 3410 NY=INT(((N-1)+NP)/V(6)+.99)
OA 3420 PRINT "ENTER THE RANGE OF YEARS YOU'D LIK
        E TO EXAMINE (FIRST, LAST)"
EC 3430 INPUT F1,L1
CO 3440 IF L1<=NY THEN 3460
OJ 3450 L1=NY
DJ 3460 FOR J1=1 TO L1
DA 3470 IF J1<F1 THEN 3490
AA 3480 GOSUB 5390
GB 3490 FOR J=1 TO V(6)-N+1
JH 3500 I=INT(P*V(3)/V(6)*100+.5)/100:B=I:GOSUB 5
        470:I$=B$
LE 3510 N5=N5+1
LB 3520 PP=V(7)-I
EE 3530 IF J1<>NY THEN 3570
BF 3540 IF N5<>NP THEN 3570
GE 3550 PP=P
HG 3560 F=1
JG 3570 IF J1<F1 THEN 3600
JN 3580 PRINT MID$(STR$(N5),2,LEN(STR$(N5))-1);TA
        B(S1);:B=P:GOSUB 5470:PRINT B$;
IM 3590 PRINT TAB(S2);:B=PP:GOSUB 5470:PRINT B$;Q
        $;TAB(S3);
GP 3600 P=P+I-V(7)
HB 3610 IF F=0 THEN 3640
KA 3620 P=0
PG 3630 J=V(6)
DJ 3640 IF J1<F1 THEN 3670
ON 3650 PRINT I$;TAB(S4);:B=P:GOSUB 5470:PRINT B$
        ;
BN 3660 PRINT
GJ 3670 I1=I1+I
DJ 3680 P1=P1+PP
DJ 3690 C5=C5+1
OK 3700 IF C5<>D5 THEN 3770
DH 3710 IF J1<F1 THEN 3770
KA 3720 GOSUB 5330
NM 3730 GOSUB 5450
EK 3740 C5=0
AA 3750 IF J=V(6)-N+1 THEN 3770
AA 3760 GOSUB 5390
IP 3770 NEXT J
JN 3780 IF J1<F1 THEN 3890
JJ 3790 IF F=0 THEN 3820
GP 3800 PRINT
FL 3810 PRINT "FINAL PAYMENT :$";INT((PP+I)*100+.
        5)/100
```

```
GF 382Ø PRINT
IK 383Ø PRINT "TOTAL INT PAID IN YR ";J1;":$";INT
        (I1*1ØØ+.5)/1ØØ
KA 384Ø PRINT "TOTAL PRINC PAID IN YR ";J1;":$";I
        NT(P1*1ØØ+.5)/1ØØ
LB 385Ø IF F=1 THEN 393Ø
HN 386Ø IF J1=L1 THEN 393Ø
LB 387Ø GOSUB 533Ø
ON 388Ø GOSUB 545Ø
FL 389Ø C5=Ø
GE 39ØØ P1=Ø
EN 391Ø I1=Ø
KC 392Ø N=1
MD 393Ø NEXT J1
DD 394Ø GOTO 276Ø
GG 395Ø C=C+1
EA 396Ø IF C<>3 THEN 399Ø
QI 397Ø PRINT V(3)*1ØØ,
LD 398Ø GOTO 4ØØØ
GN 399Ø PRINT V(C),
HE 4ØØØ A$=""
GE 4Ø1Ø INPUT A$
JJ 4Ø2Ø IF A$<>"" THEN 4Ø4Ø
AH 4Ø3Ø RETURN
QJ 4Ø4Ø IF A$<>"MR" THEN 41ØØ
IF 4Ø5Ø PRINT "MEM=";M;"  USE AS VARIABLE HERE (Y
        /N)"
HD 4Ø6Ø INPUT A$
DD 4Ø7Ø IF A$="N" THEN 4ØØØ
PC 4Ø8Ø V(C)=M
CJ 4Ø9Ø RETURN
LN 41ØØ IF A$<>"X" THEN 413Ø
PK 411Ø E=C
AG 412Ø RETURN
DI 413Ø V(C)=VAL(A$)
GP 414Ø IF C<>3 THEN 416Ø
DE 415Ø V(C)=V(C)/1ØØ
BC 416Ø RETURN
OD 417Ø REM CALCULATOR MODE
NA 418Ø GOSUB 545Ø
IK 419Ø M5=Ø
KP 42ØØ GOSUB 453Ø
HI 421Ø INPUT A$
DM 422Ø IF ASC(A$)>57 THEN 425Ø
QL 423Ø T=VAL(A$)
MP 424Ø GOTO 421Ø
HK 425Ø FOR I=1 TO 8
AF 426Ø IF A$<>MID$(V$,I,1) THEN 429Ø
OO 427Ø PRINT V(I)
QI 428Ø T=V(I)
```

```
HB 4290 NEXT I
EF 4300 FOR J=1 TO 6
JN 4310 IF A$<>MID$(C1$,(J-1)*2+1,2) THEN 4330
KG 4320 ON J GOSUB 4580,4600,4620,4640,4660,4680
HM 4330 NEXT J
EJ 4340 FOR K=1 TO 4
LG 4350 IF A$<>MID$(C$,K,1) THEN 4370
IC 4360 ON K GOSUB 4410,4460,4530,4560
ID 4370 NEXT K
QP 4380 IF M5=0 THEN 4210
IO 4390 M5=0
AG 4400 RETURN
GC 4410 FOR I=1 TO 8
PA 4420 PRINT MID$(V$,I,1);"  ";V(I)
GD 4430 NEXT I
GE 4440 PRINT
BF 4450 RETURN
KJ 4460 PRINT "IN WHAT VARIABLE ";
IO 4470 INPUT A$
HH 4480 FOR I=1 TO 8
HK 4490 IF A$<>MID$(V$,I,1) THEN 4510
BE 4500 V(I)=M
GP 4510 NEXT I
BO 4520 RETURN
EF 4530 PRINT C$;" ";C1$;" MEM=";M
AG 4540 PRINT
JH 4550 RETURN
IC 4560 M5=1
CN 4570 RETURN
FC 4580 M=M+T
HM 4590 GOTO 4690
FO 4600 M=M-T
FG 4610 GOTO 4690
EJ 4620 M=M*T
GM 4630 GOTO 4690
HM 4640 M=M/T
GC 4650 GOTO 4690
LN 4660 T=M
HI 4670 GOTO 4690
KB 4680 M=0
DA 4690 PRINT "MEM=";M
BM 4700 RETURN
CD 4710 PRINT "*FUTURE VALUE $"
FF 4720 C=0
PG 4730 GOSUB 3950
CI 4740 RETURN
NA 4750 PRINT "*PRESENT VALUE $"
HJ 4760 C=1
AC 4770 GOSUB 3950
CE 4780 RETURN
```

```
MF 4790 PRINT "PRINCIPAL $"
GJ 4800 C=1
PC 4810 GOSUB 3950
KO 4820 P=V(C)
BH 4830 RETURN
CP 4840 PRINT "ANNUAL INT RATE (%)"
HA 4850 C=2
AB 4860 GOSUB 3950
CD 4870 RETURN
EN 4880 PRINT "FOR # OF YEARS"
IE 4890 C=3
PB 4900 GOSUB 3950
BD 4910 RETURN
EK 4920 PRINT "FOR # OF MONTHS"
IM 4930 C=4
QN 4940 GOSUB 3950
BA 4950 Y=V(C-1)+V(C)/12
CC 4960 RETURN
FF 4970 PRINT "# OF PERIODS (COMPOUNDING, DEPOSIT
        S, WITHDRAWALS, PAYMENTS) YEARLY"
JD 4980 C=5
BM 4990 GOSUB 3950
QP 5000 RETURN
CA 5010 PRINT "PAYMENTS $"
II 5020 C=6
PJ 5030 GOSUB 3950
BL 5040 RETURN
FL 5050 PRINT "TERM OF LOAN:"
CL 5060 GOSUB 4880
NH 5070 GOSUB 4920
BH 5080 RETURN
HM 5090 PRINT
JA 5100 PRINT "FUTURE VALUE:$";V(1)
AE 5110 RETURN
GJ 5120 PRINT
JP 5130 PRINT "REQUIRED INVESTMENT:$";V(2)
BN 5140 RETURN
GC 5150 PRINT
PA 5160 PRINT "ANNUAL INT RATE (%) REQUIRED:";V(3
        )*100
BG 5170 RETURN
PK 5180 V(5)=V(4)-INT(V(4))
FH 5190 V(5)=INT(INT(12*V(5)*10+.5)/10)
EC 5200 V(4)=INT(V(4))
LF 5210 IF V(5)<>12 THEN 5240
CH 5220 V(4)=V(4)+1
EJ 5230 V(5)=0
GB 5240 PRINT
ME 5250 PRINT "# OF YEARS AND MONTHS:";V(4);",";V
        (5)
```

```
BF 5260 RETURN
HK 5270 PRINT
FD 5280 IF TE>=0 THEN 5310
BI 5290 PRINT "THIS IS A LOSING INVESTMENT."
AF 5300 RETURN
LK 5310 PRINT "THIS IS A PROFITABLE INVESTMENT."
BL 5320 RETURN
GA 5330 PRINT
OE 5340 PRINT "HIT <ENTER> TO CONTINUE"
JK 5350 A$=""
IK 5360 INPUT A$
BN 5370 IF A$<>"" THEN 5350
CN 5380 RETURN
OI 5390 GOSUB 5450
EG 5400 PRINT "LOAN AMORTIZATION SCHEDULE FOR YR
        ";J1
HD 5410 PRINT "PRIN $";V(2);"  RATE ";V(3)*100;"%
        ";"  PAYM $";V(7)
QP 5420 PRINT
DM 5430 PRINT "#   BEG BAL   PRINC   INT    END
        BAL"
JD 5440 RETURN
OK 5450 CLS
CJ 5460 RETURN
NK 5470 TE=0:B$=STR$(B):FOR K=1 TO LEN(B$):IF MID
        $(B$,K,1)="." THEN TE=K:K=LEN(B$)
JJ 5480 NEXT K
FJ 5490 IF TE=0 THEN RETURN ELSE B$=MID$(B$,1,TE+
        2):RETURN
```

Notemaker

Alfred J. Bruey

Good documentation can be a valuable tool for any programmer. "Notemaker" illustrates one easy way to add documentation without requiring the use of REM statements.

Documenting a program is often helpful, often necessary. First, we'll discuss the various levels of documentation. Then we'll introduce the "Notemaker" concept and an associated program. It can also be used as a quick-and-dirty word processing program (only four lines long) which you can use for your letter writing or for other quick text generation.

Documentation Levels

There are several levels of computer program documentation:

(1) *Operating instructions for the user*—a simple explanation of how to use the system. This should be written for the user who neither knows nor cares about the internals of the computer and the program. The explanations at this level will have to be very specific and should presuppose no knowledge.

(2) *More detailed user documentation*—a reference manual for the user. The assumption is that the person reading this manual is familiar with the computer operations described in the user manual. The documentation at this level might contain a detailed description of the assumptions and analyses used in solving the particular problem the system addresses. The documentation might contain flowcharts and listings of the programs. The writer might even tell the user how to modify parts of the program.

(3) *Professional documentation.* The final, most detailed, level of documentation may go into several levels of detail. It's designed for the company which wrote the program. For example, documentation required by a department manager, an analyst, and a programmer will be different. The system documentation will consist of almost every scrap of written material that was used in the design and coding of the program. It will contain all of the items in 1 and 2 above, plus whatever

else the programmer, the analyst, or management feels is necessary in case there is need to modify or make additions to the program.

Also, there have been many cases where a released program still contains errors (affectionately called bugs). If the programmer who wrote the program goes to work for another company, as has been known to happen, the documentation must be complete enough that a new programmer will be able to find bugs in a program which was written by someone else.

The Notemaker

As you can see, the "Notemaker" program is very short. It *should* be short, because it doesn't do much. It's the concept that's important, not the program. Let's look at the program a line at a time:

Line 5000 A standard usage of the INKEY$ statement. This statement is executed over and over until a key is pressed. Be sure there are no spaces between the pair of double quote marks.

Line 5010 Check to see if an *at* (@) sign has been pressed. This signals the end of the input. If the @ sign has been pressed, perform a carriage return on the printer and stop the program. If you think you'll need an @ sign in your text, you can use some other character in this line.

Line 5020 Print whatever character has been pressed both on the screen and into the printer buffer.

Line 5030 Go back for another character.

Running the Program

Running the program is simple. Just type RUN and press the Enter key. You'll get no response from the computer; it will sit there waiting for you to type in text. The text that you enter will be printed on the screen (as it usually is) and also entered into the printer buffer. Every time the printer buffer gets full or whenever you press the Enter key (or when the LPRINT in Line 5010 is executed), the text will be printed. Note that the special correction keys, such as the back arrow, will not correct what is going to the printer. If you make a mistake, it's going to show up on the printer in some way or another.

This program is to be used for taking notes, to remind yourself of something you just thought of and don't want to

forget. You could, of course, take notes with a pencil and paper but it's easier, and in most cases faster, to stay on the keyboard.

Using the Program

The program is easy to use. Just type it into the computer when you are beginning to develop a program. Make the line numbers high enough so that the coding will reside above the program you are going to write. Now that you've entered the program, you can use it in one or more of the following ways:

(1) To keep notes as you write the program. All you have to do is type RUN 5000 and press Enter. Then type in whatever notes you wish to keep and press the @ sign to signal that you're finished. Whatever you've just entered will be recorded on the printer for future reference. Figure 1 is a sample.

Figure 1. A Sample Notemaker Session

```
Notes for the data handling routine

1. Remember to explain the theory behind
    the matrix inversion technique.

2. Don't forget to re-initialize the mat-
   rices after you have completed the mat-
   rix multiplication routine.

3. The following variable names have been
   reserved and should not be used in this
   program:

              XM YZ N2 PR%
```

(2) To provide final documentation at the program level. It is often recommended that programmers insert many REM statements in a program. This is good advice which in practice is often impossible to follow because of the amount of memory that the REM statements require. The Notemaker technique will allow the programmer to simulate the use of REMs without taking up any memory space in the computer. To use the program in this way, use the following procedure:

(a) Type RUN 5000 and press Enter.
(b) Type in the desired text.
(c) Press the @ key.
(d) List the program lines to which the preceding notes applied.

This process is continued until the program has been described in sufficient detail. Figure 2 is an example.

Figure 2. Documentation Example

```
10 DIM X(10,10)
20 FOR I=1 TO 10
30 FOR J=1 TO 10
40 X(I,J)= 0
50 NEXT J
60 NEXT I
See if the user wants to read value into
 the array. If so, find array position.
Then accept data and continue until user
 signals end.
70 PRINT"DO YOU WANT TO ENTER VALUE (Y O
R N) ";
80 A$=INKEY$: IF A$="" THEN 80
85 PRINT A$
90 IF A$="N" THEN 150
100 INPUT" ENTER ROW NUMBER ";R%
110 INPUT" ENTER COLUMN NUMBER ";C%
The preceding two lines are for testing
only. In production programs, lines must
 be added to make sure that r% and c% ar
e in the range from 1 to 10.
120 PRINT" ENTER VALUE FOR ARRAY ELEMENT
 X(";R%;",";C%;")";
130 INPUT X(R%,C%)
140 GOTO 70
Next section prints out the final array
and ends the program. Come here from lin
e 90. In final program, we would need t
o add routine to save the data on disk o
r tape.

150 FOR I=1 TO 10
160 PRINT
170 FOR J=1 TO 10
180 PRINT X(I,J);
190 NEXT J
200 NEXT I
210 END
```

(3) To use the program as a mini–word processor, just enter the program and type RUN. Then type your letter. You have to press Return when you see that you're getting near the end of the line, or this program will break a word in the middle whether you want it to or not. Also, there's no way to save what you've entered or make corrections or move para-

graphs or perform any other of the well-known word processor functions, but the program is quick to enter and easy to use. Figure 3 is an example.

Figure 3. Word Processing Example

```
Dear Reader,

Here is an example of a short note which
 can be written using the four line prog
ram described in thsi article. There is
no way to correct an error like the one
I made in the sentence above.

              Sincerely,
              The Author
```

The Notemaker Program

```
5000 A$=INKEY$: IF A$="" THEN 5000
5010 IF A$="@" THEN LPRINT CHR$(13):LPRINT:STO
     P
5020 LPRINT A$;:PRINT A$
5030 GOTO 5000
```

Personalized Form Letters

Donald B. Trivette

If you've ever needed to mail copies of the same letter to a number of people—for holiday greetings, notices of club meetings, or whatever—you'll appreciate this labor-saving program. It automatically retrieves addresses and salutations from disk and prints them atop your form letter. The program requires an IBM PC or PCjr with BASICA or Cartridge BASIC, a disk drive, and a printer. A word processor that saves standard ASCII files is recommended.

'Tis the season to be jolly. 'Tis also the season to send out holiday cards and letters. You remember Christmas letters, those mimeographed missives that let your archfriends know how well you're doing—or how well you want them to *think* you're doing. Perhaps you've not participated in this holiday ritual because it's just too much trouble to duplicate and address 50 letters—and besides, mimeographed letters are so impersonal.

Now, with the assistance of your IBM PC or PCjr, you too can practice creative writing. The BASIC program following this article automatically merges an address list with a letter to produce a *personalized* form letter. It's guaranteed to speed up your holiday correspondence and leave your recipients wondering whether they've been form-lettered or not.

Of course, "Personalized Form Letters" isn't limited to holiday greetings. You might use this program to contact everyone in the neighborhood about the proposed zoning change to put a nuclear waste dump adjacent to the playground, or to keep the members of the garden club or user group informed about the next meeting. If you occasionally need to send the same letter to many people, and don't want to invest in a commercial form-letter program, then read on.

Standard ASCII Files

Personalized Form Letters is only 76 lines long (53 if you leave out the comments at the beginning). It uses the input from two files, files that you must create using a word proces-

sor, a text editor, or the DOS utility program EDLIN. However the files are created, they must be standard ASCII text.

One file contains an exact image of the letter. This means that if you're using a word processor to create the letter, you must *not* count on it to format the lines, insert spaces, and adjust the right margin. Instead, *you* must decide how many characters to put on each line of the letter; you must format it manually. If your word processor automatically wraps words from one line to another, as most do, you'll need to defeat that feature. For example, text with 50 characters on a line is about right for standard margins, so when a line of text reaches column 50, press the Enter key and start the next line. In other words, type the letter just as you would on an old-fashioned typewriter.

Personalized Form Letters is a dumb program. It won't understand the special codes that switch on boldface printing, underlining, centering, or any of the fancy things your word processor can do. It just reads a line from a file and prints it.

But it's not completely stupid, either. It does know enough to print one letter for each address in the address file. How do you signal the computer where to put the address? Insert <<>> at the proper location in the letter and the program will replace it with a four-line address, a blank line, the salutation, and another blank line. For example:

700 Maple Avenue
Anywhere, NC 27900
December 10, 1984

<<>>

Hi. We've had a wonderful year . . . .
Made so much money that we don't know
how we'll ever spend it . . . .

By inserting a few blank lines ahead of your own address, you can position the letter so the recipient's address appears through a window envelope when the paper is folded. The program automatically reprints the first letter until you get it properly aligned. (Maybe you can find red window envelopes for the holidays.)

The Address List

The second ASCII file required by the program contains the address list. Again, you may use a word processor to build and maintain the file. Remember to press the Enter key after each line in the address. Personalized Form Letters is designed to use a four-line address and a one-line salutation. The salutation—*Dear Bob & Ann*—adds a personal touch. Insert a blank line between each address/salutation group. That's to make it easier for you to separate one address from another when editing the address file. Here's an example of how two addresses would look:

Mr. and Mrs. Bob Adams
123 Main Street
Westover, NH 93939

Dear Bob and Ann,

Dr. and Mrs. Robert Brown
Apartment 203
7000 Southfork Avenue
Snake Bluff, CO 94959

Dear Bob & Carol & Ted & Alice,

Notice that the Adams' address is only three lines long, so a blank line is entered as the fourth line of their address.

Personalized Form Letters is designed to print on continuous-form paper. Who wants to feed in 50 sheets one at a time? You do? Then insert these two lines in the program:

```
374 PRINT "Insert paper and press any key."
375 B$=INKEY$:IF B$="" THEN 375
```

and it will pause after printing each letter.

Type the BASIC program exactly as it's shown. Save it. Then create your letter and address files as described above. Next, return to BASIC and run the program with those files as input. One important point: You must use Advanced BASIC (BASICA) or PCjr Cartridge BASIC when running this program (ordinary BASIC will result in a SYNTAX ERROR in line 560).

C H A P T E R 3

Personalized Form Letters

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
IL 10 REM IBM Personalized Form Letters
GB 20 REM
DI 30 REM A program to print form letters using
QL 40 REM addresses from an address file with
BE 50 REM the following format:
NF 60 REM Address line 1
PL 70 REM Address line 2
AB 80 REM Address line 3
BH 90 REM Address line 4
HC 100 REM Salutation
FG 110 REM (blank line to separate one
JI 120 REM address from another)
QK 130 REM
LO 140 REM The letter file is an ASCII file
MD 150 REM containing the form letter.
AA 160 REM
IN 170 REM Use <<>> to indicate where the
HI 180 REM address/salutation is to appear in the
FC 190 REM letter. The program automatically
GN 200 REM inserts a blank line before and after
MC 210 REM the salutation.
QJ 220 REM
CA 230 REM -------------
NH 240 KEY OFF:CLS
FJ 250 ON ERROR GOTO 730
JN 260 PRINT
FL 270 PRINT"IBM Personalized Form Letters"
JB 280 PRINT
MD 290 LINE INPUT "Enter address filename: ";ADD$
IM 300 LINE INPUT "Enter letter filename : ";LETR
       $
DL 310 LINE INPUT "Enter left margin value: ";N$
GI 320 N=VAL(N$)
NE 330 I=0
II 340 CLOSE #2:OPEN ADD$ FOR INPUT AS #2
JM 350 CLOSE #1:OPEN LETR$ FOR INPUT AS #1
DH 360 IF I<2 THEN GOSUB 580
PF 370 LPRINT CHR$(12)  'skip to top of page
NK 380 IF EOF(1) THEN GOTO 350
QP 390 LINE INPUT #1, A$
CN 400 IF A$="<<>>" THEN GOSUB 440  'print addres
       s
ML 410 LPRINT SPC(N)A$
HP 420 GOTO 380
FB 430 REM ---GOSUB to print address---
BL 440 I=I+1          'count of letters
OE 450 FOR J=1 TO 4  '4-line address
```

```
HG 460 IF EOF(2) THEN PRINT:PRINT I-1;" Letters p
       rinted.":END
FI 470 LINE INPUT #2,A$
AG 480 LPRINT SPC(N)A$   'print on printer
EK 490 PRINT A$          'print on screen
NF 500 NEXT J
MH 510 LPRINT:PRINT
MK 520 LINE INPUT #2,A$   'salutation
MA 530 LPRINT SPC(N)A$
KE 535 PRINT A$           'print to screen
NN 540 LPRINT:PRINT
KI 550 LINE INPUT #2, A$ 'throw away blank line
AH 560 RETURN 380
KO 570 REM ---GOSUB to line up letter---
EN 580 IF I<>0 THEN GOTO 630
MN 590 PRINT "Switch on printer and press any key
        to continue."
IF 600 PRINT
MJ 610 B$=INKEY$:IF B$="" THEN GOTO 610
ME 620 RETURN
LI 630 LPRINT CHR$(12)
GH 640 PRINT STRING$(48,"*")
PC 650 PRINT "*  Is the letter properly aligned (
       Y/N/Esc) ?  *"
GI 660 PRINT STRING$(48,"*"):PRINT:PRINT:LOCATE ,
       ,0
BN 670 B$=INKEY$:IF B$="" THEN 670
GO 680 IF B$=CHR$(27) THEN END
KL 690 IF B$="Y" OR B$="y" THEN RETURN
OF 700 IF B$="N" OR B$="n" THEN PRINT "Make adjus
       tments...":RETURN 310
KG 710 BEEP:GOTO 670
HP 720 REM ---ERRORS---
FP 730 IF ERR=53 AND ERL=340 THEN PRINT "Address
       file not found.":END
PB 740 IF ERR=53 AND ERL=350 THEN PRINT "Letter f
       ile not found.":END
ON 750 ON ERROR GOTO 0
MM 760 END
```

Calendar Maker

Paul C. Liu
Translation by Kevin Mykytyn

Have you ever wished you had a calendar for the coming year? "Calendar Maker" can generate precise monthly, yearly, and appointment calendars for the next five centuries. It runs on both the PC and PCjr and requires a printer.

There are times when you need to check a date or plan an event a year or more in advance. It may be a vacation, a business trip, or a special anniversary. What would be a good week next February to cruise the Caribbean? How will that 1987 convention you're supposed to attend fit in with your vacation plans? Will your birthday in 1995 fall on a weekday? Will your wedding anniversary in 1988 coincide with a weekend?

Unfortunately, it's not that easy to lay your hands on a calendar for the coming year, much less calendars for years far in the future. That's where "Calendar Maker" comes in handy.

With Calendar Maker and a printer, you can generate four kinds of calendars: a monthly calendar on your video screen; a large, printed monthly wall calendar; a printed monthly appointment calendar; and a large, printed yearly calendar. The program flawlessly calculates these calendars for any month or year in the next five centuries—provided, of course, that the current calendar system is not revised (it was last changed in 1752).

Making Calendars

Calendar Maker consists of four BASIC programs. Each makes a different type of calendar. All the programs, when run, ask you for relevant information, such as the month and year of the calendar you wish to make. The numbers you enter should be separated by a comma, and years should be the full four digits (1986, not 86).

Program 1 displays a monthly calendar on your monitor screen.

Program 2 prints out a large monthly calendar for hanging on a wall (see Figure 1). You can, of course, generate a smaller monthly calendar by using Program 1 and pressing the PrtSc (Print Screen) key.

Program 3 also prints out a monthly calendar, but in a different format. It tabulates the days of the month vertically as a list. This is intended to be an appointment calendar for your desk, with room for short notes each day. It is labeled with the month and year (Figure 2).

Program 4 prints out a full yearly calendar with the message "Happy New Year" at the top. Of course, you can easily modify the program to print any message you want (Figure 3).

After you enter the month and year as requested, Programs 2, 3, and 4 prompt you to bring the printer online if it is offline or switched off. When using tractor-feed paper, remember to align the perforation with the printhead before switching on the printer so the calendar will fit on a single page.

Note: These programs were written for the IBM Graphics Printer (Epson MX-80), but should work with little or no modification on a wide variety of printers.

Calendar Calculating

You may be wondering how these programs can accurately calculate days and dates so far into the future. The key is an algorithm which, with minor adjustments for leap years and leap centuries, systematically figures out which day falls on which date of which year.

If we let D1 be the day of the week (for Sunday D1=1, for Monday D1=2, and so on), and let M, D, and Y be the month, day, and year, D1 can be calculated in BASIC like this:

```
D1=INT(2.6*(M-2)-0.2)+D+Y-1900+INT((Y-1900)/4)
D1=D1+INT(19/4)-2*19
D1=D1-INT(D1/7)*7+1
```

This formula requires two modifications. If M is equal to 1 or 2, we must add 12 and subtract Y by 1. In other words, we consider the months January and February as the thirteenth and fourteenth months of the previous year. Also, if M is equal to 4 or 9, the calculated value for D1 must be increased by 1.

The resulting formula works perfectly for the twentieth and twenty-first centuries, up to the year 2100. Beyond that,

you run into problems with leap centuries and nonleap centuries. Century years such as 2100 and 2200 which are not divisible by 400 are not leap years.

To correct the formula for this, D1 must be reduced by 1 after March 2100, and again every 100 years afterward. Calendar Maker contains such modifications to insure accuracy over the next 500 years.

Once we know the day of the week for the given date, especially the first day of the month, the rest of the calendar-maker task is just a matter of setting up and preparing the proper format and display.

Figure 1. Wall Calendar

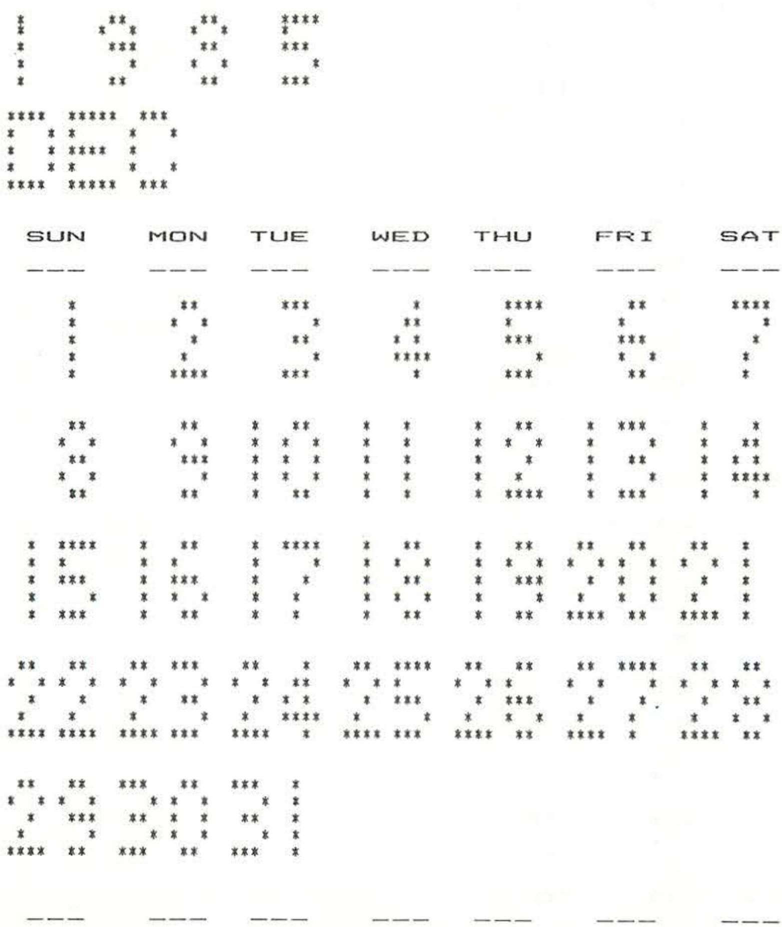


Figure 2. Appointment Calendar

```
    JANUARY  1986

 WEDNESDAY   1   (  1   )
  THURSDAY   2   (  2   )
    FRIDAY   3   (  3   )
  SATURDAY   4   (  4   )
    SUNDAY   5   (  5   )

    MONDAY   6   (  6   )
   TUESDAY   7   (  7   )
 WEDNESDAY   8   (  8   )
  THURSDAY   9   (  9   )
    FRIDAY  1Ø   (  1Ø   )
  SATURDAY  11   (  11   )
    SUNDAY  12   (  12   )

    MONDAY  13   (  13   )
   TUESDAY  14   (  14   )
 WEDNESDAY  15   (  15   )
  THURSDAY  16   (  16   )
    FRIDAY  17   (  17   )
  SATURDAY  18   (  18   )
    SUNDAY  19   (  19   )

    MONDAY  2Ø   (  2Ø   )
   TUESDAY  21   (  21   )
 WEDNESDAY  22   (  22   )
  THURSDAY  23   (  23   )
    FRIDAY  24   (  24   )
  SATURDAY  25   (  25   )
    SUNDAY  26   (  26   )

    MONDAY  27   (  27   )
   TUESDAY  28   (  28   )
 WEDNESDAY  29   (  29   )
  THURSDAY  3Ø   (  3Ø   )
    FRIDAY  31   (  31   )
```

Figure 3. Yearly Calendar

HAPPY NEW YEAR 1987

JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

MARCH

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

APRIL

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

MAY

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

JUNE

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

JULY

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

AUGUST

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

SEPTEMBER

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

OCTOBER

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

NOVEMBER

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

DECEMBER

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Program 1. Screen Calendar

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
EL 5 KEY OFF:WIDTH 40
KG 10 D$=CHR$(31):U$=CHR$(30):R$=CHR$(28):L$=CHR$
      (29):H$=CHR$(11):TB$=CHR$(223):SP$=CHR$(32)
LL 80 DIM M$(12):FOR I=1 TO 12:READ M$(I):NEXT I
FN 100 CLS:PRINT D$D$D$R$R$"THIS IS A PROGRAM":PR
       INT R$R$R$R$R$R$ "TO SHOW A"
PC 105 COLOR 6:PRINT R$R$R$"MONTHLY CALENDAR":COL
       OR 2:PRINT R$R$R$R$"ON THE SCREEN"
QN 110 PRINT D$D$R$R$ "PLEASE TYPE IN THE":COLOR
       4:PRINT R$R$R$ "MONTH";: COLOR 2:PRINT " A
       ND ";:COLOR 4:PRINT "YEAR":COLOR 2
MD 111 PRINT R$ "THAT YOU WISH TO SEE":PRINT D$R$
       R$ "(EXAMPLE:";:COLOR 4:PRINT "12,1983";:C
       OLOR 2:PRINT ")";:PRINT D$D$
FD 130 PRINT R$R$R$R$R$;:INPUT M0,Y:COLOR 6:PRINT
        R$R$R$R$R$D$D$ "THANK YOU!":COLOR 2:PRINT
        D$D$: FOR I= 1 TO 800:NEXT
EA 292 IF M0=1 OR M0=3 OR M0=5 OR M0=7 OR M0=8 OR
        M0=10 OR M0=12 THEN E1=31
FP 293 IF M0=4 OR M0=6 OR M0=9 OR M0=11 THEN E1=3
       0
GN 294 IF M0=2 AND Y/4<>INT(Y/4) THEN E1=28
PP 295 IF M0=2 AND Y/4=INT(Y/4) THEN GOSUB 1400
HI 297 CLS:COLOR 4:PRINT D$R$ M$ (M0)" " Y ;:COLO
       R 2
NK 298 GOSUB 1350:IF A=40 THEN PRINT
JA 300 COLOR 4:PRINT R$R$ "S";:COLOR 2:PRINT "  M
         T  W  T  F  S"
FL 305 GOSUB 1360
JO 310 D=1:GOSUB 1050
MH 320 IF D1=7 THEN FOR I= 1 TO 19:PRINT R$;:NEXT
       :PRINT D:IF A=40  THEN PRINT
BB 321 IF D1=7 THEN 330
JF 322 IF D1=6 THEN FOR I =1 TO 16:PRINT R$;:NEXT
       :PRINT D;:GOTO 330
BD 323 IF D1=5 THEN FOR I= 1 TO 13:PRINT R$;:NEXT
       :PRINT D;:GOTO 330
NL 324 IF D1=4 THEN FOR I =1 TO 10:PRINT R$;:NEXT
       :PRINT D;:GOTO 330
IH 325 IF D1=3 THEN FOR I =1 TO 7 :PRINT R$;:NEXT
       :PRINT D;:GOTO 330
CF 326 IF D1=2 THEN FOR I =1 TO 4 :PRINT R$;:NEXT
       :PRINT D;:GOTO 330
OI 327 IF D1=1 THEN PRINT R$;:COLOR 4:PRINT D;:CO
       LOR 2:GOTO 330
DN 330 FOR D= 2 TO E1:GOSUB 1050
```

```
NE 331 IF D1=1 AND D<=9 THEN COLOR 4:PRINT R$ D;:
       COLOR 2:GOTO 345
DP 332 IF D1=1 AND D> 9 THEN COLOR 4:PRINT D;:COL
       OR 2:GOTO 345
DM 333 IF D1=7 THEN 34Ø
FK 334 IF D<=9 THEN PRINT D;:GOTO 345
JL 335 PRINT L$ D;:GOTO 345
OD 34Ø IF D>9 THEN PRINT L$ D:GOTO 345
MJ 341 PRINT D
HJ 345 IF A=4Ø AND D1=7 THEN PRINT
LB 346 NEXT D
MN 1Ø45 PRINT:PRINT:FL=1:GOSUB 135Ø:IF A=22 THEN
        PRINT U$U$U$
CM 1Ø49 PRINT D$:END
ML 1Ø5Ø IF MØ=1 THEN MØ=13:Y=Y-1:GOTO 1Ø8Ø
HG 1Ø6Ø IF MØ=2 THEN MØ=14:Y=Y-1
BC 1Ø8Ø M=MØ-2
GH 11ØØ D1=INT(2.6*M-.2)+D+Y-19ØØ+INT((Y-19ØØ)/4)
PG 115Ø D1=D1+INT (19/4)-2*19
KE 12ØØ D1=D1-INT(D1/7)*7+1
AF 121Ø IF MØ=24 OR MØ=9 THEN D1=D1+1
DG 123Ø IF MØ=13 THEN MØ=1 :Y=Y+1:GOTO 1245
IK 124Ø IF MØ=14 THEN MØ=2:Y=Y+1:D1=D1+1
BD 1244 IF D1=8 THEN D1=1
KB 1245 IF (Y=21ØØ AND MØ>=3) OR (Y>21ØØ) THEN D1
        =D1-1:IF D1=Ø THEN D1=7
OJ 1247 IF (Y=22ØØ AND MØ>=3) OR (Y>22ØØ) THEN D1
        =D1-1:IF D1=Ø THEN D1=7
BB 1249 IF (Y=23ØØ AND MØ>=3) OR (Y>23ØØ) THEN D1
        =D1-1:IF D1=Ø THEN D1=7
JO 125Ø RETURN
LL 135Ø IF FL=Ø THEN PRINT :FOR I= 1 TO 22:PRINT
        "*";:NEXT :PRINT:RETURN
PD 1355 IF D1=7 THEN PRINT U$U$U$ :FOR I=1 TO 22:
        PRINT "*";:NEXT:PRINT U$:RETURN
PO 1358 FOR I=1 TO 22:PRINT "*";:NEXT:PRINT U$:RE
        TURN
DI 136Ø FOR QQ=1 TO 7:PRINT SP$SP$TB$;:NEXT:PRINT
        :RETURN
PP 14ØØ IF (Y/1ØØ=INT(Y/1ØØ)) AND (Y/4ØØ<>INT(Y/4
        ØØ)) THEN E1=28:GOTO 141Ø
HG 14Ø5 E1=29
IG 141Ø RETURN
DM 142Ø DATA "   JANUARY","   FEBRUARY","     MARC
        H","     APRIL"
GJ 143Ø DATA "       MAY","       JUNE","       JULY","
            AUGUST"
HB 144Ø DATA "  SEPTEMBER","   OCTOBER","    NOVEM
        BER","   DECEMBER"
```

Program 2. Wall Calendar

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KP 1 KEY OFF:WIDTH 40:GOTO 20
QM 5 E1=1:E2=1:E3=1:E4=1:E5=1:E6=1:E7=1
GM 6 GOSUB 1109:D8=D7-1:RETURN
JO 20 GOSUB 4000:GOSUB 3200:LPRINT ""
LH 30 ON M0 GOSUB 3010,3020,3030,3040,3050,3060,3
      070,3080,3090,3100,3110,3120
OM 40 LPRINT "":LPRINT "":GOSUB 1610:GOSUB 1650:G
      OSUB 1660
DN 80 ON D9 GOSUB 1811,1821,1831,1841,1851,1861,1
      871
CJ 99 LPRINT "":LPRINT ""
PN 100 G1=D8
DM 105 G=G1:GOSUB 1720:D1=D:E1=E
OI 110 G2=G+1:G=G2:GOSUB 1720:D2=D:E2=E
EL 115 G3=G+1:G=G3:GOSUB 1720:D3=D:E3=E
IC 120 G4=G+1:G=G4:GOSUB 1720:D4=D:E4=E
OF 125 G5=G+1:G=G5:GOSUB 1720:D5=D:E5=E
DM 130 G6=G+1:G=G6:GOSUB 1720:D6=D:E6=E
JP 135 G7=G+1:G=G7:GOSUB 1720:D7=D:E7=E
OO 140 G1=G7+1:GOSUB 1109:LPRINT "":LPRINT "":IF
       G1<=E9 THEN 105
FE 155 LPRINT ""
LH 1000 GOTO  5000
EO 1109 GOSUB 2000:X=E1:X1=D1:GOSUB 11000
MA 1120 X=E2:X1=D2:GOSUB 11000
OL 1130 X=E3:X1=D3:GOSUB 11000
PG 1140 X=E4:X1=D4:GOSUB 11000
BB 1150 X=E5:X1=D5:GOSUB 11000
DM 1160 X=E6:X1=D6:GOSUB 11000
EM 1170 X=E7:X1=D7:FL=1:GOSUB 11000
GD 1209 GOSUB 2000:X=E1:X1=D1:GOSUB 12000
OK 1220 X=E2:X1=D2:GOSUB 12000
PF 1230 X=E3:X1=D3:GOSUB 12000
BA 1240 X=E4:X1=D4:GOSUB 12000
DL 1250 X=E5:X1=D5:GOSUB 12000
EG 1260 X=E6:X1=D6:GOSUB 12000
GL 1270 X=E7:X1=D7:FL=1:GOSUB 12000
JI 1309 GOSUB 2000:X=E1:X1=D1:GOSUB 13000
PE 1320 X=E2:X1=D2:GOSUB 13000
BP 1330 X=E3:X1=D3:GOSUB 13000
DK 1340 X=E4:X1=D4:GOSUB 13000
EF 1350 X=E5:X1=D5:GOSUB 13000
GA 1360 X=E6:X1=D6:GOSUB 13000
IK 1370 X=E7:X1=D7:FL=1:GOSUB 13000
LN 1409 GOSUB 2000:X=E1:X1=D1:GOSUB 14000
BO 1420 X=E2:X1=D2:GOSUB 14000
DJ 1430 X=E3:X1=D3:GOSUB 14000
```

```
EE 144Ø X=E4:X1=D4:GOSUB 14ØØØ
GP 145Ø X=E5:X1=D5:GOSUB 14ØØØ
IK 146Ø X=E6:X1=D6:GOSUB 14ØØØ
KJ 147Ø X=E7:X1=D7:FL=1:GOSUB 14ØØØ
NC 15Ø9 GOSUB 2ØØØ:X=E1:X1=D1:GOSUB 15ØØØ
DI 152Ø X=E2:X1=D2:GOSUB 15ØØØ
ED 153Ø X=E3:X1=D3:GOSUB 15ØØØ
GO 154Ø X=E4:X1=D4:GOSUB 15ØØØ
IJ 155Ø X=E5:X1=D5:GOSUB 15ØØØ
JE 156Ø X=E6:X1=D6:GOSUB 15ØØØ
MI 157Ø X=E7:X1=D7:FL=1:GOSUB 15ØØØ
IH 16ØØ RETURN
BO 161Ø LPRINT "      ";:LPRINT CHR$(14)"SUN";:LP
        RINT CHR$(3Ø) "   ";
CF 1611 LPRINT CHR$(14)"MON";:LPRINT CHR$(3Ø) "
        ";
LC 1612 LPRINT CHR$(14)"TUE";:LPRINT CHR$(3Ø) "
         ";
DD 1613 LPRINT CHR$(14)"WED";:LPRINT CHR$(3Ø) "
        ";
BP 1614 LPRINT CHR$(14)"THU";:LPRINT CHR$(3Ø) "
         ";
KF 1615 LPRINT CHR$(14)"FRI";:LPRINT CHR$(3Ø) "
         ";
LP 1616 LPRINT CHR$(14)"SAT":LPRINT CHR$(3Ø) " "
MO 162Ø LPRINT "      ";:LPRINT CHR$(14)"---";:LP
        RINT CHR$(3Ø) "   ";
EC 1621 LPRINT CHR$(14)"---";:LPRINT CHR$(3Ø) "
        ";
ID 1622 LPRINT CHR$(14)"---";:LPRINT CHR$(3Ø) "
         ";
FK 1623 LPRINT CHR$(14)"---";:LPRINT CHR$(3Ø) "
        ";
JL 1624 LPRINT CHR$(14)"---";:LPRINT CHR$(3Ø) "
         ";
JP 1625 LPRINT CHR$(14)"---";:LPRINT CHR$(3Ø) "
         ";
EA 1626 LPRINT CHR$(14)"---":LPRINT CHR$(3Ø) " ":
        RETURN
LP 165Ø IF MØ=1 OR MØ=3 OR MØ=5 OR MØ=7 OR MØ=8 O
        R MØ=1Ø OR MØ=12 THEN E9=31
GC 1652 IF MØ=4 OR MØ=6 OR MØ=9 OR MØ=11 THEN E9=
        3Ø
OB 1654 IF MØ=2 AND Y/4<>INT(Y/4) THEN E9=28
EG 1656 IF MØ=2 AND Y/4=INT(Y/4) THEN E9=29
LG 1658 RETURN
NA 166Ø IF MØ=1 THEN MØ=13:Y=Y-1:GOTO  167Ø
JG 1665 IF MØ=2 THEN MØ=14:Y=Y-1
CL 167Ø M=MØ-2
NC 1675 D9=INT(2.6*M-.2)+D+Y-19ØØ+INT((Y-19ØØ)/4)
```

```
KA 1680 D9=D9+INT(19/4)-2*19
PI 1685 D9=D9-INT(D9/7)*7+1
BN 1690 IF M0=24 OR M0=9 THEN D9=D9+1
FN 1695 IF M0=13 THEN M0=1:Y=Y+1:GOTO  1710
JA 1700 IF M0=14 THEN M0=2:Y=Y+1:D9=D9+1
AF 1705 IF D9=8 THEN D9=1
GG 1710 IF (Y=2100 AND M0>=3) OR (Y>2100) THEN D9
        =D9-1:IF D9=0 THEN D9=7
KK 1711 IF (Y=2200 AND M0>=3) OR (Y>2200) THEN D9
        =D9-1:IF D9=0 THEN D9=7
NO 1712 IF (Y=2300 AND M0>=3) OR (Y>2300) THEN D9
        =D9-1:IF D9=0 THEN D9=7
KA 1715 RETURN
BL 1720 IF G>E9 THEN GOTO  1740
PP 1722 IF G<10 THEN GOTO  1742
DJ 1726 IF G>=10 AND G<20 THEN GOTO  1746
KO 1728 IF G>=20 AND G<30 THEN GOTO  1748
LO 1730 IF G>=30 THEN GOTO  1750
FJ 1740 D=1:E=1:GOTO  1755
GM 1742 D=G+2:E=1:GOTO  1755
IJ 1746 D=G-10+2:E=2:GOTO  1755
KM 1748 D=G-20+2:E=3:GOTO  1755
MF 1750 D=G-30+2:E=4
LM 1755 RETURN
DH 1811 D1=1:D2=3:D3=4:D4=5:D5=6:D6=7:D7=8:GOSUB
         5:RETURN
JN 1821 D1=1:D2=1:D3=3:D4=4:D5=5:D6=6:D7=7:GOSUB
         5:RETURN
PM 1831 D1=1:D2=1:D3=1:D4=3:D5=4:D6=5:D7=6:GOSUB
         5:RETURN
GJ 1841 D1=1:D2=1:D3=1:D4=1:D5=3:D6=4:D7=5:GOSUB
         5:RETURN
OJ 1851 D1=1:D2=1:D3=1:D4=1:D5=1:D6=3:D7=4:GOSUB
         5:RETURN
HB 1861 D1=1:D2=1:D3=1:D4=1:D5=1:D6=1:D7=3:GOSUB
         5:RETURN
QK 1871 D1=3:D2=4:D3=5:D4=6:D5=7:D6=8:D7=9:GOSUB
         5:RETURN
KE 2000 LPRINT "    ";:RETURN
OO 2001 LPRINT " ** ";:RETURN
OC 2002 LPRINT "*  *";:RETURN
OG 2003 LPRINT "*  *";:RETURN
PK 2004 LPRINT "*  *";:RETURN
PO 2005 LPRINT " ** ";:RETURN
EB 2011 LPRINT " *  ";:RETURN
EF 2012 LPRINT " *  ";:RETURN
FJ 2013 LPRINT " *  ";:RETURN
FN 2014 LPRINT " *  ";:RETURN
FB 2015 LPRINT " *  ";:RETURN
OE 2021 LPRINT " ** ";:RETURN
```

```
PI 2022 LPRINT "*  *";:RETURN
FG 2023 LPRINT "  * ";:RETURN
FA 2024 LPRINT " *  ";:RETURN
DK 2025 LPRINT "****";:RETURN
HD 2031 LPRINT "*** ";:RETURN
GP 2032 LPRINT "   *";:RETURN
PP 2033 LPRINT " ** ";:RETURN
GH 2034 LPRINT "   *";:RETURN
ID 2035 LPRINT "*** ";:RETURN
FE 2041 LPRINT "  * ";:RETURN
PO 2042 LPRINT " ** ";:RETURN
PI 2043 LPRINT "* * ";:RETURN
DM 2044 LPRINT "****";:RETURN
GE 2045 LPRINT "  * ";:RETURN
CD 2051 LPRINT "****";:RETURN
EH 2052 LPRINT "*   ";:RETURN
IB 2053 LPRINT "*** ";:RETURN
HN 2054 LPRINT "   *";:RETURN
JJ 2055 LPRINT "*** ";:RETURN
PA 2061 LPRINT " ** ";:RETURN
FK 2062 LPRINT "*   ";:RETURN
IE 2063 LPRINT "*** ";:RETURN
OM 2064 LPRINT "*  *";:RETURN
AA 2065 LPRINT " ** ";:RETURN
DJ 2071 LPRINT "****";:RETURN
HL 2072 LPRINT "   *";:RETURN
GF 2073 LPRINT "  * ";:RETURN
GP 2074 LPRINT " *  ";:RETURN
GD 2075 LPRINT " *  ";:RETURN
PG 2081 LPRINT " ** ";:RETURN
OK 2082 LPRINT "*  *";:RETURN
OO 2083 LPRINT " ** ";:RETURN
AC 2084 LPRINT "*  *";:RETURN
AG 2085 LPRINT " ** ";:RETURN
OJ 2091 LPRINT " ** ";:RETURN
ON 2092 LPRINT "*  *";:RETURN
LL 2093 LPRINT " ***";:RETURN
IJ 2094 LPRINT "   *";:RETURN
BJ 2095 LPRINT " ** ";:RETURN
FN 2111 LPRINT "  * ";:RETURN
FB 2112 LPRINT "  * ";:RETURN
FF 2113 LPRINT "  * ";:RETURN
GJ 2114 LPRINT "  * ";:RETURN
GN 2115 LPRINT "  * ";:RETURN
KG 3010 GOSUB 2000:LPRINT " ***   ***  *   *"
MA 3011 GOSUB 2000:LPRINT "  *   *   * **   *"
NO 3012 GOSUB 2000:LPRINT "  *   *   * * * *"
LE 3013 GOSUB 2000:LPRINT "* *   ***** *  **"
FO 3014 GOSUB 2000:LPRINT "***   *   * *   *"
JE 3015 RETURN
```

```
FB 3020 GOSUB 2000:LPRINT "***** ***** **** "
NF 3021 GOSUB 2000:LPRINT "*     *     *   *"
MB 3022 GOSUB 2000:LPRINT "***   ****  **** "
ON 3023 GOSUB 2000:LPRINT "*     *     *   *"
BF 3024 GOSUB 2000:LPRINT "*     ***** **** "
JH 3025 RETURN
JG 3030 GOSUB 2000:LPRINT "*   *  ***  **** "
HE 3031 GOSUB 2000:LPRINT "** ** *   * *   *"
GC 3032 GOSUB 2000:LPRINT "* * * *   * **** "
EC 3033 GOSUB 2000:LPRINT "* * * ***** *  * "
GE 3034 GOSUB 2000:LPRINT "*   * *   * *   *"
KK 3035 RETURN
ON 3040 GOSUB 2000:LPRINT " ***  ****  **** "
GL 3041 GOSUB 2000:LPRINT "*   * *   * *   *"
NH 3042 GOSUB 2000:LPRINT "*   * ****  **** "
CF 3043 GOSUB 2000:LPRINT "***** *     *  * "
BJ 3044 GOSUB 2000:LPRINT "*   * *     *   *"
KN 3045 RETURN
KE 3050 GOSUB 2000:LPRINT "*   *  ***  *   *"
IK 3051 GOSUB 2000:LPRINT "** ** *   * *   *"
HA 3052 GOSUB 2000:LPRINT "* * * *   *  * * "
NM 3053 GOSUB 2000:LPRINT "* * * *****   *  "
OE 3054 GOSUB 2000:LPRINT "*   * *   *   *  "
KA 3055 RETURN
HL 3060 GOSUB 2000:LPRINT " ***  *   * *   *"
NP 3061 GOSUB 2000:LPRINT "  *   *   * **  *"
ON 3062 GOSUB 2000:LPRINT "  *   *   * * * *"
OF 3063 GOSUB 2000:LPRINT "* *   *   * *  **"
KH 3064 GOSUB 2000:LPRINT "***    ***  *   *"
KD 3065 RETURN
NE 3070 GOSUB 2000:LPRINT " ***  *   * *    "
MM 3071 GOSUB 2000:LPRINT "  *   *   * *    "
MA 3072 GOSUB 2000:LPRINT "  *   *   * *    "
MO 3073 GOSUB 2000:LPRINT "* *   *   * *    "
EM 3074 GOSUB 2000:LPRINT "***    ***  *****"
KG 3075 RETURN
PH 3080 GOSUB 2000:LPRINT " ***  *   *  *** "
NN 3081 GOSUB 2000:LPRINT "*   * *   * *    "
QL 3082 GOSUB 2000:LPRINT "*   * *   * *  **"
KJ 3083 GOSUB 2000:LPRINT "***** *   * *   *"
ED 3084 GOSUB 2000:LPRINT "*   *  ***   *** "
LJ 3085 RETURN
HM 3090 GOSUB 2000:LPRINT " **** ***** **** "
PK 3091 GOSUB 2000:LPRINT "*     *     *   *"
PE 3092 GOSUB 2000:LPRINT " ***  ****  **** "
IA 3093 GOSUB 2000:LPRINT "    * *     *    "
MC 3094 GOSUB 2000:LPRINT "****  ***** *    "
LM 3095 RETURN
```

```
DH 3100 GOSUB 2000:LPRINT " ***   ***  *****"
NL 3101 GOSUB 2000:LPRINT "*   * *   *   *  "
HB 3102 GOSUB 2000:LPRINT "*   * *       *  "
ND 3103 GOSUB 2000:LPRINT "*   * *   *   *  "
DP 3104 GOSUB 2000:LPRINT " ***   ***    *  "
JD 3105 RETURN
KK 3110 GOSUB 2000:LPRINT "*   *  ***  *   *"
GI 3111 GOSUB 2000:LPRINT "**  * *   * *   *"
GG 3112 GOSUB 2000:LPRINT "* * * *   * *   *"
HE 3113 GOSUB 2000:LPRINT "*  ** *   *  * * "
CE 3114 GOSUB 2000:LPRINT "*   *  ***    *  "
JG 3115 RETURN
MP 3120 GOSUB 2000:LPRINT "****  *****  *** "
QJ 3121 GOSUB 2000:LPRINT "*   * *     *   *"
EB 3122 GOSUB 2000:LPRINT "*   * ****  *    "
AB 3123 GOSUB 2000:LPRINT "*   * *     *   *"
NP 3124 GOSUB 2000:LPRINT "****  *****  *** "
KJ 3125 RETURN
HB 3200 I1=INT(Y/1000):J1=Y-I1*1000:I2=INT(J1/100
        ):J2=J1-I2*100:I3=INT(J2/10)
EP 3210 I4=J2-I3*10
KK 3211 IF I2=0 THEN I2=10
MM 3212 IF I3=0 THEN I3=10
OO 3213 IF I4=0 THEN I4=10
CH 3214 GOSUB 2000:X=I1:GOSUB 6000:GOSUB 2000:X=I
        2:GOSUB 6000:GOSUB 2000:X=I3:GOSUB 6000
JA 3215 GOSUB 2000:X=I4:FL=1:GOSUB 6000
NO 3314 GOSUB 2000:X=I1:GOSUB 7000:GOSUB 2000:X=I
        2:GOSUB 7000:GOSUB 2000:X=I3:GOSUB 7000
LD 3315 GOSUB 2000:X=I4:FL=1:GOSUB 7000
HF 3414 GOSUB 2000:X=I1:GOSUB 8000:GOSUB 2000:X=I
        2:GOSUB 8000:GOSUB 2000:X=I3:GOSUB 8000
NG 3415 GOSUB 2000:X=I4:FL=1:GOSUB 8000
CM 3514 GOSUB 2000:X=I1:GOSUB 9000:GOSUB 2000:X=I
        2:GOSUB 9000:GOSUB 2000:X=I3:GOSUB 9000
QJ 3515 GOSUB 2000:X=I4:FL=1:GOSUB 9000
IJ 3614 GOSUB 2000:X=I1:GOSUB 10000:GOSUB 2000:X=
        I2:GOSUB 10000:GOSUB 2000:X=I3
FH 3615 GOSUB 10000:GOSUB 2000:X=I4:FL=1:GOSUB 10
        000:RETURN
JD 4000 CLS:PRINT "THIS IS A PROGRAM":PRINT "  TO
         PRINT A"
JL 4020 PRINT "MONTHLY CALENDAR":PRINT " ON THE P
        RINTER"
BD 4030 PRINT:PRINT "PLEASE TYPE IN THE":PRINT "
         MONTH AND YEAR"
JP 4035 PRINT "THAT YOU WISH TO SEE":PRINT " (EXA
        MPLE: 12,1983)":PRINT TAB(3);
EO 4060 INPUT M0,Y
```

```
KB 4080 PRINT "THANK YOU! NOW--":PRINT " PLEASE T
        URN ON THE"
NM 4085 PRINT "PRINTER AND  then  TYPE":PRINT TAB
        (8)"OK":INPUT R$
AG 4110 IF R$<>"OK" AND R$<>"ok" THEN 4080
PJ 4130 PRINT "PRINTING":FOR I=1 TO 800:NEXT :RET
        URN
CJ 4999 LPRINT CHR$(30) " "
HH 5000 GOSUB 1620
HJ 5001 END
BI 6000 ON X GOSUB 2011,2021,2031,2041,2051,2061,
        2071,2081,2091,2001
KG 6010 IF FL<>1 THEN LPRINT " ";:RETURN
LB 6020 LPRINT "":FL=0:RETURN
LC 7000 ON X GOSUB 2012,2022,2032,2042,2052,2062,
        2072,2082,2092,2002
KH 7010 IF FL<>1 THEN LPRINT " ";:RETURN
LC 7020 LPRINT "":FL=0:RETURN
GM 8000 ON X GOSUB 2013,2023,2033,2043,2053,2063,
        2073,2083,2093,2003
LI 8010 IF FL<>1 THEN LPRINT " ";:RETURN
LD 8020 LPRINT "":FL=0:RETURN
AG 9000 ON X GOSUB 2014,2024,2034,2044,2054,2064,
        2074,2084,2094,2004
LJ 9010 IF FL<>1 THEN LPRINT " ";:RETURN
LE 9020 LPRINT "":FL=0:RETURN
BL 10000 ON X GOSUB 2015,2025,2035,2045,2055,2065
         ,2075,2085,2095,2005
MP 10010 IF FL<>1 THEN LPRINT " ";:RETURN
FO 10020 LPRINT "":FL=0:RETURN
PO 11000 ON X GOSUB 2000,2111,2021,2031:LPRINT "
         ";
IF 11010 ON X1 GOSUB 2000,2001,2011,2021,2031,204
         1,2051,2061,2071,2081,2091
QM 11020 IF FL<>1 THEN LPRINT "  ";:RETURN
HN 11030 FL=0:LPRINT "":RETURN
FN 12000 ON X GOSUB 2000,2112,2022,2032:LPRINT "
         ";
HG 12010 ON X1 GOSUB 2000,2002,2012,2022,2032,204
         2,2052,2062,2072,2082,2092
QO 12020 IF FL<>1 THEN LPRINT "  ";:RETURN
HP 12030 FL=0:LPRINT "":RETURN
LM 13000 ON X GOSUB 2000,2113,2023,2033:LPRINT "
         ";
GH 13010 ON X1 GOSUB 2000,2003,2013,2023,2033,204
         3,2053,2063,2073,2083,2093
AA 13020 IF FL<>1 THEN LPRINT "  ";:RETURN
HB 13030 FL=0:LPRINT "":RETURN
BL 14000 ON X GOSUB 2000,2114,2024,2034:LPRINT "
         ";
```

```
GI 14Ø1Ø ON X1 GOSUB 2ØØØ,2ØØ4,2Ø14,2Ø24,2Ø34,2Ø4
         4,2Ø54,2Ø64,2Ø74,2Ø84,2Ø94
AC 14Ø2Ø IF FL<>1 THEN LPRINT "   ";:RETURN
HD 14Ø3Ø FL=Ø:LPRINT "":RETURN
HK 15ØØØ ON X GOSUB 2ØØØ,2115,2Ø25,2Ø35:LPRINT "
         ";
FJ 15Ø1Ø ON X1 GOSUB 2ØØØ,2ØØ5,2Ø15,2Ø25,2Ø35,2Ø4
         5,2Ø55,2Ø65,2Ø75,2Ø85,2Ø95
AE 15Ø2Ø IF FL<>1 THEN LPRINT "   ";:RETURN
HF 15Ø3Ø FL=Ø:LPRINT "":RETURN
```

Program 3. Appointment Calendar

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KO 8Ø DIM M$(12),W$(7):FOR I=1 TO 12:READ M$(I):N
      EXT I:FOR I=1 TO 7:READ W$(I):NEXT I
NN 1ØØ CLS:PRINT D$SP$SP$ "THIS IS A  PROGRAM":PR
       INT R$R$R$R$R$R$ "     TO SHOW A"
GO 1Ø5 COLOR 6:PRINT R$R$ " MONTHLY CALENDAR":COL
       OR 2:PRINT R$R$R$ " ON THE PRINTER":PRINT
       D$
KJ 11Ø PRINT R$ "PLEASE TYPE IN THE":COLOR 4:PRIN
       T R$R$R$ "  MONTH";:COLOR 2:PRINT " AND ";
       :COLOR 4:PRINT "YEAR":COLOR 2
HF 111 PRINT "THAT YOU WISH TO SEE":PRINT R$ "(EX
       AMPLE 12,1983";:COLOR 2:PRINT ")";D$D$:COL
       OR 6
FC 12Ø PRINT TAB(5);:INPUT MØ,Y
GN 13Ø COLOR 2:PRINT D$D$SP$SP$ "THANK YOU! NOW--
       ":PRINT" PLEASE";:COLOR 6:PRINT " TURN ON
       ";:COLOR 2:PRINT" THE"
BK 131 PRINT "PRINTER AND THEN TYPE":PRINT TAB(9)
       " ";:COLOR 6:PRINT "OK" D$:INPUT R$
BN 151 IF R$<>"OK" AND R$<>"ok" THEN 13Ø
AB 154 PRINT "PRINTING" D$:FOR I=1 TO 8ØØ:NEXT:GO
       SUB 1292
HA 2Ø2 LPRINT CHR$(14) SP$SP$SP$ M$(MØ) SP$ Y:GOS
       UB 16ØØ:GOSUB 17ØØ:FOR D=1 TO E1:J1=J1 +1
NH 21Ø GOSUB 1Ø5Ø:IF D<1Ø THEN G$=" "
OH 213 IF D>=1Ø THEN G$=""
PM 214 IF D1=1 THEN LPRINT CHR$(3Ø) SP$SP$SP$ W$(
       D1) CHR$(14) G$ D CHR$(3Ø) "(" J1 ")":GOSU
       B 16ØØ
GD 215 IF D1=1 THEN GOSUB 16ØØ
ED 217 IF D1=1 THEN GOTO 22Ø
BB 219 LPRINT CHR$(3Ø) SP$SP$SP$ W$(D1) CHR$(14)
       G$ D CHR$(3Ø) "(" J1 ")":GOSUB 16ØØ
KK 22Ø NEXT D
GB 1ØØØ END
```

```
IE 1050 IF M0=1 THEN M0=13: Y=Y-1:GOTO 1080
HG 1060 IF M0=2 THEN M0=14:Y=Y-1
BC 1080 M=M0-2
GH 1100 D1=INT(2.6*M-.2)+D+Y-1900+INT((Y-1900)/4)
PG 1150 D1=D1+INT (19/4)-2*19
KE 1200 D1=D1-INT(D1/7)*7+1
AF 1210 IF M0=24 OR M0=9 THEN D1=D1+1
DG 1230 IF M0=13 THEN M0=1 :Y=Y+1:GOTO 1245
IK 1240 IF M0=14 THEN M0=2:Y=Y+1:D1=D1+1
BD 1244 IF D1=8 THEN D1=1
KB 1245 IF (Y=2100 AND M0>=3) OR (Y>2100) THEN D1
        =D1-1:IF D1=0 THEN D1=7
OJ 1247 IF (Y=2200 AND M0>=3) OR (Y>2200) THEN D1
        =D1-1:IF D1=0 THEN D1=7
BB 1249 IF (Y=2300 AND M0>=3) OR (Y>2300) THEN D1
        =D1-1:IF D1=0 THEN D1=7
JO 1250 RETURN
JL 1292 IF M0=1 OR M0=3 OR M0=5 OR M0=7 OR M0=8 O
        R M0=10 OR M0=12 THEN E1=31
AC 1293 IF M0=4 OR M0=6 OR M0=9 OR M0=11 THEN E1=
        30
NH 1294 IF M0=2 AND Y/4 <> INT(Y/4) THEN E1=28
HE 1295 IF M0=2 AND Y/4 =  INT(Y/4) THEN GOSUB 14
        00
LC 1296 RETURN
DD 1400 IF (Y/100=INT(Y/100))AND (Y/400<>INT (Y/4
        00)) THEN E1=28:GOTO 1410
HG 1405 E1=29
IG 1410 RETURN
QP 1600 FOR I=1 TO 20:LPRINT CHR$(30)  SP$;:NEXT
        I
NO 1605 FOR K=1 TO 18:LPRINT ".  ";:NEXT K:LPRINT
         "."
JK 1610 RETURN
FA 1700 IF M0=1 THEN J1=0
BN 1702 IF M0=2 THEN J1=31
AF 1704 IF M0=3 THEN J1=59
KC 1706 IF M0=4 THEN J1=90
LA 1707 IF M0=5 THEN J1=120
CB 1709 IF M0=6 THEN J1=151
FF 1711 IF M0=7 THEN J1=181
PG 1713 IF M0=8 THEN J1=212
GH 1715 IF M0=9 THEN J1=243
IH 1717 IF M0=10 THEN J1=273
CE 1719 IF M0=11 THEN J1=304
GM 1721 IF M0=12 THEN J1=334
IE 1723 IF Y/4<>INT(Y/4) THEN 1730
ME 1725 IF (Y/100=INT(Y/100)) AND (Y/400<>INT(Y/4
        00)) THEN 1730
```

```
JC 1727 IF (Y/4=INT(Y/4)) AND (MØ>=3) THEN J1=J1+
        1
JC 173Ø RETURN
KP 2ØØØ DATA "  JANUARY"," FEBRUARY","    MARCH",
        "    APRIL","      MAY"
KK 2Ø1Ø DATA "     JUNE","     JULY","   AUGUST",
        "SEPTEMBER","  OCTOBER"
EF 2Ø2Ø DATA " NOVEMBER"," DECEMBER"
EL 2Ø3Ø DATA "   SUNDAY","   MONDAY","  TUESDA
        Y","WEDNESDAY"," THURSDAY"
ML 2Ø4Ø DATA "   FRIDAY"," SATURDAY"
```

Program 4. Yearly Calendar

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
ND 1 R$=CHR$(2Ø):DW$=CHR$(14):D$=CHR$(31):SP$=CHR
     $(32):Q$ =CHR$(28)
QK 2 WIDTH "LPT1:",81
DI 5 DIM W4(3):GOSUB 151Ø:I=1:J=2
JF 7 LPRINT DW$ SPC(13) "HAPPY NEW YEAR ";Y:LPRIN
     T
NK 1Ø LPRINT DW$ SPC(8) "JANUARY" SPC(13) "FEBRUA
      RY"
LJ 12 GOSUB 1ØØ9:GOSUB 1ØØØ:GOSUB 1Ø12:CØ=6:GOSUB
       1Ø19:GOSUB 1ØØØ:GOSUB 1Ø22
KL 15 MØ=I:M8=1:GOSUB 292:GOSUB 2Ø:GOTO 35
CC 2Ø D=1:GOSUB 1Ø5Ø:W2=8-D1:W4(M8)=W2+1:GOSUB 32
      1
QK 22 IF D1=7 THEN 3Ø
OE 25 FOR D=2 TO W2:GOSUB 1Ø5Ø:GOSUB 331:NEXT D
DN 3Ø RETURN
OL 35 GOSUB 99Ø:MØ=J:M8=2:GOSUB 292:GOSUB 2Ø
CM 44 W3=1
OM 45 MØ=I:M8=1:GOSUB 292:GOSUB 2ØØ
DA 46 IF W4(2)=9 THEN LPRINT R$ SPC(1);
MP 5Ø GOSUB 991:MØ=J:M8=2:GOSUB 292:GOSUB 2ØØ
FH 56 IF W3=1 AND W4(1)>9 THEN LPRINT R$ SPC(Ø);
KL 57 IF W3=1 AND W4(1)<1Ø THEN LPRINT R$ SPC(1);
QI 58 IF W3=4 AND W4(2)>3Ø THEN LPRINT R$ SPC(Ø);
DP 65 W3=W3+1
FC 7Ø IF W3<CØ THEN 45
QI 71 LPRINT " "
NE 72 IF I=1 THEN 86
QI 73 IF I=3 THEN 96
PC 74 IF I=5 THEN 1Ø6
BH 75 IF I=7 THEN 116
EM 76 IF I=9 THEN 126
KP 77 IF I=11 THEN 199
DK 86 LPRINT DW$ SPC(9) "MARCH" SPC(16) "APRIL"
```

```
HF 88 I=3:J=4:GOTO 12
JP 96 LPRINT DW$ SPC(1Ø) "MAY" SPC(17) "JUNE"
JG 98 I=5:J=6:GOTO 12
BM 1Ø6 LPRINT DW$ SPC(9) "JULY" SPC(16) "AUGUST"
BC 1Ø8 I=7:J=8:GOTO 12
FF 116 LPRINT DW$ SPC(7) "SEPTEMBER" SPC(13) "OCT
       OBER"
BL 118 I=9:J=1Ø:GOTO 12
LD 126 LPRINT DW$ SPC(7) "NOVEMBER" SPC(13) "DECE
       MBER"
JP 128 I=11:J=12:GOTO 12
EM 199 LPRINT R$ SPC(1) :END
NG 2ØØ D4=W4(M8):D7=W4(M8)+6
GP 2Ø5 D=D4:GOSUB 1Ø5Ø
JK 21Ø IF D1<>1 THEN PRINT "WHY D1=";D1
GB 212 IF M8=1 AND (D+1)<1Ø THEN GOSUB 528
LG 213 IF M8=1 AND (D+1)>9 THEN GOSUB 53Ø
FN 214 IF M8=2 AND (D+1)<1Ø THEN GOSUB 428
MP 215 IF M8=2 AND D4>=3Ø AND D4<=E1 THEN GOSUB 4
       33:GOTO 217
QP 216 IF M8=2 AND (D+1)>9 THEN GOSUB 43Ø
DB 217 FOR D=D4+1 TO D7:GOSUB 1Ø5Ø:GOSUB 331:NEXT
HG 22Ø W4(M8)=D7+1
NP 225 RETURN
EA 292 IF MØ=1 OR MØ=3 OR MØ=5 OR MØ=7 OR MØ=8 OR
        MØ=1Ø OR MØ=12 THEN E1=31
FP 293 IF MØ=4 OR MØ=6 OR MØ=9 OR MØ=11 THEN E1=3
       Ø
EP 294 IF MØ=2 AND Y/4 <> INT(Y/4) THEN E1=28
LA 295 IF MØ=2 AND Y/4 =  INT(Y/4) THEN GOSUB 14Ø
       Ø
OA 296 RETURN
PI 321 IF D1=7 THEN LPRINT R$ SPC(36) D;:GOTO 33Ø
EL 322 IF D1=6 THEN LPRINT R$ SPC(31) D;:GOTO 33Ø
MI 323 IF D1=5 THEN LPRINT R$ SPC(26) D;:GOTO 33Ø
BL 324 IF D1=4 THEN LPRINT R$ SPC(21) D;:GOTO 33Ø
JI 325 IF D1=3 THEN LPRINT R$ SPC(16) D;:GOTO 33Ø
OL 326 IF D1=2 THEN LPRINT R$ SPC(11) D;:GOTO 33Ø
DB 327 IF D1=1 THEN LPRINT R$ SPC(6) D;:GOTO 33Ø
AB 328 LPRINT R$ SPC(3) D;:GOTO 33Ø
EF 329 LPRINT R$ SPC(2) D;
MD 33Ø RETURN
MG 331 IF D>E1 THEN LPRINT R$ SPC(5);:GOTO 35Ø
BF 332 IF D1=1 AND D<=9 THEN LPRINT D;:GOTO 35Ø
HK 333 IF D1=1 AND D>9 THEN LPRINT D;:GOTO 35Ø
HB 335 IF D<=9  THEN LPRINT R$ SPC(2) D;:GOTO 35Ø
DL 336 LPRINT R$ SPC(1) D;
MH 35Ø RETURN
AG 428 IF D>E1 THEN LPRINT R$ SPC(9);:GOTO 435
LI 429 GOTO 328
```

```
PA 430 IF D>E1 THEN LPRINT R$ SPC(9);:GOTO 435
KO 431 GOTO 329
DG 433 LPRINT R$ SPC(2) D;
ND 435 RETURN
DA 528 IF D>E1 THEN LPRINT R$ SPC(9);:GOTO 535
IA 529 GOTO 532
CK 530 IF D>E1 THEN LPRINT R$ SPC(9);:GOTO 535
HG 531 GOTO 533
BI 532 LPRINT R$ SPC(6) D;:GOTO 535
HA 533 LPRINT R$ SPC(5) D;
NE 535 RETURN
FP 990 LPRINT R$ SPC(3);:GOTO 992
KP 991 LPRINT R$ SPC(6);
OL 992 RETURN
LN 1000 LPRINT R$ SPC(7);
IP 1001 RETURN
IB 1009 LPRINT R$ SPC(3);
EJ 1010 LPRINT "    S    M    T    W    T    F
         S";
IC 1011 RETURN
ML 1012 LPRINT "    S    M    T    W    T    F
         S"
JK 1013 RETURN
IE 1019 LPRINT R$ SPC(3);
GA 1020 LPRINT "    -    -    -    -    -    -
         -" ;
IF 1021 RETURN
GH 1022 LPRINT "    -    -    -    -    -    -
         -"
JN 1023 RETURN
ML 1050 IF M0=1 THEN M0=13:Y=Y-1:GOTO 1080
HG 1060 IF M0=2 THEN M0=14:Y=Y-1
BC 1080 M=M0-2
GH 1100 D1=INT(2.6*M-.2)+D+Y-1900+INT((Y-1900)/4)
AF 1150 D1=D1+INT(19/4)-2*19
KE 1200 D1=D1-INT(D1/7)*7+1
AF 1210 IF M0=24 OR M0=9 THEN D1=D1+1
LF 1230 IF M0=13 THEN M0=1:Y=Y+1:GOTO 1250
IK 1240 IF M0=14 THEN M0=2:Y=Y+1:D1=D1+1
BD 1244 IF D1=8 THEN D1=1
AE 1245 IF (Y=2100 AND M0>3) OR (Y>2100) THEN D1=
        D1-1:IF  D1=0 THEN D1=7
EL 1247 IF (Y=2200 AND M0>3) OR (Y>2200) THEN D1=
        D1-1:IF  D1=0 THEN D1=7
HC 1249 IF (Y=2300 AND M0>3) OR (Y>2300) THEN D1=
        D1-1:IF  D1=0 THEN D1=7
JO 1250 RETURN
DD 1400 IF (Y/100=INT(Y/100))AND (Y/400<>INT (Y/4
        00)) THEN E1=28:GOTO 1410
HG 1405 E1=29
```

```
IG 1410 RETURN
JL 1510 CLS:PRINT D$Q$Q$ "THIS IS A PROGRAM":PRIN
        T Q$Q$Q$Q$Q$Q$ "TO SHOW A"
LF 1520 COLOR 6:PRINT Q$Q$Q$ "YEARLY CALENDAR":CO
        LOR 2:PRINT Q$Q$Q$ "ON THE PRINTER":PRINT
         D$
MM 1530 PRINT Q$ "PLEASE TYPE IN THE":PRINT Q$Q$Q
        $ "YEAR THAT YOU":PRINT Q$Q$Q$Q$ "WISH TO
         SEE"
JB 1535 PRINT Q$Q$Q$ "(EXAMPLE:";:COLOR 6:PRINT "
        1984";:COLOR 2:PRINT ")" D$D$:PRINT TAB(6
        );:INPUT Y
CJ 1570 PRINT D$Q$Q$Q$ "THANK YOU! NOW-":PRINT Q$
         "PLEASE TURN ON THE"
OH 1573 PRINT "PRINTER AND THEN TYPE"
HK 1575 COLOR 6:PRINT TAB(8) "OK" D$:COLOR 2
NO 1580 INPUT AR$
PA 1585 IF AR$<>"OK" AND AR$<>"ok" THEN 1570
PE 1590 PRINT "PRINTING" D$:FOR I=1 TO 800:NEXT:R
        ETURN
```

CHAPTER 4

Education

Munchmath

Gerald R. Anderson
Translation by Jeff Hamdani

Do you know any youngsters who need to practice their arithmetic? "Munchmath" is a math drill program which entertains as it teaches. Multiple difficulty levels make it suitable for a wide range of ages. The program runs on any PC or PCjr.

To keep a young person's interest, an educational program should be fun to play. At the very least it shouldn't be boring. "Munchmath" presents a happy-face character who relies on the player's correct answers to math problems to stay ahead of a pursuer that is trying to gobble him up.

The program begins by asking for the player's name, the type of problems wanted (addition, subtraction, multiplication, or division), and the starting level of difficulty. Problems are then displayed on the screen for the player to answer. Each correct answer scores ten points and moves "Munchie" one step closer to the power prize. The other character, however, stays in hot pursuit only three steps behind.

Making Munchie Move

After 15 correct responses, Munchie eats the power prize and the tables are turned. Munchie chases the character across the screen, eventually catching him and scoring a bonus of 100 points. The difficulty level then advances one notch higher and new problems are presented.

The pursuing character moves into action when the player gives a wrong answer. First, the correct answer is displayed for the player to study. Then the character advances one step closer to Munchie. Three incorrect answers and the character catches Munchie and gobbles him up. This results in a loss of 50 points and a return to the next lower level of difficulty.

If a Q is typed in response to a problem instead of a number, the game stops. A scoreboard is printed which shows the number of problems the player has been given, the number answered correctly, the number answered incorrectly, and the

percentage of correct answers. The player may then choose to resume the game or to end play.

This program has been extensively tested by my six- and eight-year-old daughters, as well as the neighborhood children, and its appeal holds up very nicely.

Munchmath

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
JH 100 RANDOMIZE(1):WIDTH 40:SCREEN 0,1:KEY OFF:D
       EF SEG=0:POKE 1047,80
LM 110 SM=9:L=1:COLOR 14,1,1:CLS
BA 120 H$=CHR$(11):D$=STRING$(21,31):S$=H$+D$
OE 130 J$=STRING$(22,223):P=4:M=1:GOTO 760
NM 140 D=VAL(AN$):IF ASC(AN$)=81 AND W+R>0 THEN 6
       60
OJ 150 IF INT(D) <> INT(C) THEN 180
CC 160 P=P+1:R=R+1:M=M+1:SC=SC+10
PF 170 SOUND 252,3:GOTO 560
CE 180 M=M+1:W=W+1:RW=22-LEN(C$):RV=RW-1:IF Q=3 T
       HEN LOCATE 9,12:PRINT STRING$(30,32):LOCAT
       E 10,RV:PRINT C;:FOR I=1 TO 400:NEXT I:GOT
       O 200
IE 190 LOCATE 10,12:PRINT STRING$(20,32):LOCATE 1
       0,RW-1:PRINT C;:FOR I=1 TO 400:NEXT I
BK 200 SOUND 100,.7:GOTO 580
BN 210 SOUND 200,1:FOR I=1 TO 100:NEXT I:RETURN
KE 220 RANDOMIZE(RD)
JH 230 A=INT(RND(1)*5*L)+1
CH 240 B=INT(RND(1)*5*L)+1:IF B>A THEN A=A+B
GK 250 E=INT(A*B):A$=STR$(A):B$=STR$(B):E$=STR$(E
       )
IL 260 IF Q=1 THEN C=A+B:X=43:GOTO 300
OD 270 IF Q=2 THEN C=A-B:X=45:GOTO 300
FB 280 IF Q=3 THEN C=A:GOTO 340
KG 290 C=E:X=88
PL 300 C$=STR$(C):LOCATE 7,INT((40-LEN(A$))/2):PR
       INT A
FD 310 LOCATE 8,INT((40-LEN(B$))/2):PRINT B:D$=ST
       RING$(3,223):LC=INT((40-LEN(D$))/2):LOCATE
        9,LC:PRINT D$:LOCATE 8,LC:PRINT CHR$(X)
BF 320 PRINT LEFT$(S$,10);SPC(19-LEN(C$));STRING$
       (15,32):GOSUB 970:IF AN$="" THEN 320
LF 330 D=VAL(AN$):GOTO 140
JJ 340 C$=STR$(C):LC=22-LEN(C$)
BJ 350 LE=LEN(E$):LOCATE 11,11:PRINT STRING$(20,3
       2):LOCATE 12,11:PRINT STRING$(20,32)
GG 360 LOCATE 11,LC-1:PRINT STRING$(LE,220);:IF L
       EN(B$)=3 THEN LB=2 ELSE LB=1
```

```
MO 370 LOCATE 12,LC-3-LB:PRINT B:LOCATE 12,LC-1:P
       RINT E:LOCATE 12,LC-2:PRINT CHR$(251)
LF 380 GOSUB 970:IF AN$="" THEN 380
EM 390 GOTO 140
HJ 400 LOCATE 2,SM+P:PRINT CHR$(64):FOR I=200 TO
       220:SOUND I*2,1:NEXT I
JP 410 LOCATE 2,SM+P:PRINT CHR$(32):SC=SC-50:IF S
       C<0 THEN SC=0
BA 420 L=L-1:IF L=0 THEN L=1
FA 430 P=4:M=1:CLS:GOTO 950
IL 440 LOCATE 2,SM+M-1:PRINT CHR$(32):FOR I=1 TO
       6:LOCATE 2,SM+M:PRINT CHR$(8)
AA 450 LOCATE 2,SM+P:PRINT CHR$(8):LOCATE 2,SM+P:
       PRINT CHR$(2):LOCATE 2,SM+M:PRINT CHR$(8)
MI 460 LOCATE 2,SM+M:PRINT CHR$(32):LOCATE 2,SM+P
       :PRINT CHR$(32):P=P-1:M=M-1:NEXT I
CL 470 FOR I=12 TO 9 STEP-1:LOCATE 2,SM+I:PRINT C
       HR$(8):LOCATE 2,SM+9:PRINT CHR$(8)
PN 480 SOUND INT(RND(1)*90+50),.5:LOCATE 2,SM+I:P
       RINT CHR$(2)
JL 490 LOCATE 2,SM+I:PRINT CHR$(32):NEXT I
LM 500 COLOR 4,1:FOR I=1 TO 20:PRINT CHR$(11);CHR
       $(31);TAB(15);"** 100 **";:SOUND INT(RND(0
       )*200+50),.5
ML 510 Z$=SPACE$(9):PRINT CHR$(11);CHR$(31);TAB(1
       5);Z$:NEXT I:SOUND 37,0:COLOR 7,1:L=L+1:IF
        L>9 THEN 660
KF 520 SC=SC+100:P=4:M=1
ED 530 REM SETUP
JG 540 CLS:PRINT LEFT$(S$,3);SPC(9);J$
MF 550 PRINT CHR$(11);SPC(13);CHR$(31);STRING$(14
       ,254):COLOR 15,1:LOCATE 2,28:PRINT CHR$(3)
       :COLOR 7,0
HJ 560 COLOR 14,1:LOCATE 2,SM+P-1:PRINT CHR$(32):
       LOCATE 2,SM+P:PRINT CHR$(2):COLOR 7,1
OD 570 IF SM+P=SM+19 THEN 440
IC 580 COLOR 4,1:LOCATE 2,SM+M-1:PRINT CHR$(32):L
       OCATE 2,SM+M:PRINT CHR$(8):COLOR 7,1
BJ 590 IF M=P THEN 400
BM 600 PRINT LEFT$(S$,16);SPC(16);"Level:";L
HG 610 PRINT LEFT$(S$,18);SPC(9);J$
DE 620 PRINT LEFT$(S$,21);SPC(12);N$;"'s score:";
       SC
ND 630 IF Q=3 THEN LOCATE 10,5:PRINT STRING$(20,3
       2);
HE 640 PRINT LEFT$(S$,7);SPC(17);"         ":PRINT S
       PC(17);"         ":PRINT SPC(13);"
        "
DL 650 GOTO 220
```

```
FA 66Ø CLS:PRINT CHR$(31);SPC(13-LEN(N$)/2);N$;"'
       s Scoreboard"
JJ 67Ø PRINT SPC(14);STRING$(2,31);"Problems:";W+
       R:PR=W+R
IK 68Ø PRINT SPC(12);STRING$(2,31);"Right Answers
       :";R:PRINT SPC(12);STRING$(2,31);"Wrong An
       swers:";W
LB 69Ø PRINT SPC(14);STRING$(2,31);"Grade:";INT(R
       /PR*1ØØ);"%"
BM 7ØØ PRINT SPC(12);STRING$(2,31);"Play Again (Y
       /N)?"
IP 71Ø A$=INKEY$:RD=RND(1):IF A$="" OR (A$<>"Y" A
       ND A$<>"N") THEN 71Ø
GB 72Ø IF A$="N" THEN END
GG 73Ø IF A$="Y" AND L>9 THEN RUN 1ØØ
PL 74Ø PR=Ø:R=Ø:W=Ø:SC=Ø:P=4:M=1:GOTO 85Ø
IP 75Ø REM TITLE
HN 76Ø LOCATE 1,1,Ø:FOR I=13 TO 29 STEP 2:READ CR
       ,CL:COLOR CL,1:LOCATE 1Ø,I:PRINT CHR$(CR):
       NEXT I:COLOR 14,1
FL 77Ø READ F,G:IF F=-1 THEN 8ØØ
KC 78Ø SOUND F+G,.5
GL 79Ø RD=RND(1):GOTO 77Ø
QN 8ØØ FOR I=4 TO 24:SOUND I*1Ø,.5:PRINT LEFT$(S$
       ,1Ø);SPC(I);CHR$(8);STRING$(2,32);CHR$(2);
KD 81Ø PRINT LEFT$(S$,1Ø);SPC(I);CHR$(8);STRING$(
       2,32);CHR$(2)
OK 82Ø NEXT I:SOUND 37,Ø
EN 83Ø PRINT LEFT$(S$,1Ø);SPC(24);STRING$(8,32)
BG 84Ø COLOR 7,1:CLS:LOCATE 12,11:PRINT"What is y
       our name";:BEEP:INPUT N$:IF LEN(N$)>1Ø THE
       N 84Ø
FA 85Ø COLOR 7,1:CLS:LOCATE 3,13:PRINT"What would
        you":GOSUB 21Ø
KL 86Ø LOCATE 5,12:PRINT"like to practice,":GOSUB
        21Ø
NG 87Ø LOCATE 7,INT(2Ø-LEN(N$)/2):PRINT N$;":":GO
       SUB 21Ø
EP 88Ø LOCATE 1Ø,14:PRINT CHR$(31);"1) Addition":
       GOSUB 21Ø
PF 89Ø PRINT SPC(13);CHR$(31);"2) Subtraction":GO
       SUB 21Ø
QO 9ØØ PRINT SPC(13);CHR$(31);"3) Division":GOSUB
        21Ø
NM 91Ø PRINT SPC(13);CHR$(31);"4) Multiplication"
       :GOSUB 21Ø
OM 92Ø A$=INKEY$:Q=VAL(A$):Z=RND(1):IF Q<1 OR Q>4
        THEN 92Ø
OG 93Ø PRINT LEFT$(S$,2Ø);SPC(13);"Level (1-9)?"
GD 94Ø A$=INKEY$:L=VAL(A$):IF L<1 OR L>9 THEN 94Ø
```

```
GC 950 GOTO 540
BJ 970 AN$="":RW=22-LEN(C$)
QL 980 ZA$=INKEY$:IF ZA$="" THEN 980
OC 990 ZL=LEN(AN$)
MC 1000 IF ASC(ZA$)=8 AND ZL THEN LOCATE 10,RW-1:
        PRINT CHR$(32);:AN$=LEFT$(AN$,ZL-1):RW=RW
        -1:GOTO 980
EJ 1010 IF ZA$=CHR$(13) THEN PRINT:RETURN
KF 1020 IF ZA$<>"Q" AND (ZA$<"0" OR ZA$>"9") OR Z
        L=LEN(C$)-1 THEN 980
II 1030 LOCATE 10,RW:PRINT ZA$;:RW=RW+1:AN$=AN$+Z
        A$:GOTO 980
GA 1040 DATA 77,2,85,3,78,4,67,5,72,6,77,7,65,8,8
        4,10,72,11
PH 1050 DATA 16,195,22,96,28,49,33,125,33,125,33,
        125,33,125
NG 1060 DATA 28,49,28,49,28,49,22,96,28,49,22,96,
        16,195,-1,0
```

States and Capitals Tutor

Enoch L. Moser
Translation by Tim Victor

Is Huntington the capital of West Virginia? Or is it Charleston? Then again, Charleston could be the capital of South Carolina...or is that Columbia? "States and Capitals Tutor" is not only a useful tool for students who are learning the American states and capitals, but it also demonstrates the use of arrays in BASIC programs. It runs on any PC or PCjr.

At one time or another almost everyone is stuck with the tedious assignment of memorizing all the U.S. states and capital cities. Usually, this means hours of looking over maps and geography books, and recruiting somebody else to drill you on which city is the capital of which state.

"States and Capitals Tutor" is designed to make this task a little easier. It's an educational drill program that repeatedly (and patiently) tests a student's knowledge on the subject, and keeps track of correct and incorrect responses. The program randomly decides whether to quiz the student on states or capitals and then chooses the questions randomly.

Questions answered correctly are not repeated. However, the program *will* repeat questions that are missed. And like any good teacher, States and Capitals Tutor will help students who ask for it. Students who are stumped can simply type HELP. The program gives the correct answer and comes back to the troublesome question later. It also keeps track of how many times the student asks for help.

Spelling Is Checked

In addition, States and Capitals Tutor encourages students to learn how to *spell* the place names. If a student's response is misspelled, the program cannot match it with the right answer and therefore interprets the response as a wrong answer. In other words, MISSISSIPI isn't good enough. Be careful, though, to spell the names correctly when you type in the program. Otherwise, a student would have to misspell the state or capital exactly as you did in order to get the answer

right. (Incidentally, the program accepts two spellings for the capital of Minnesota—St. Paul and Saint Paul.)

The beep sounds in the program can be toggled on or off by answering a question with the symbols + or − . The variable MS (Make Sound) is set to either 1 (yes) or 0 (no) in line 80 to control the sound. Just answer a question by typing the + or − key and press Enter. Then continue to answer the question as usual.

When all 50 states have been correctly matched with their capitals, and if the student has not asked for help or missed any questions, he or she is rewarded with a perfect score message.

To use States and Capitals Tutor, enter the program listing, save it on disk or tape, and type RUN. The screen instructions are self-explanatory.

A BASIC Tutorial

If you're interested in learning BASIC programming, States and Capitals Tutor employs a very useful technique known as *arrays*. By studying the listing you can adapt this common technique to programs of your own.

Notice how the names of all the states and capitals are included in the program's DATA statements. When the program is first run, there is a short pause of about a second before the first screen appears. During this pause, the program is storing the names of the states and capitals into a *two-dimensional array*, written as ST$(I,J). We'll explain this in a moment.

When the student gives a correct answer, the range of the random number generator is decreased by one, and that state/capital is moved to the top part of the list, out of the range of selection. Otherwise, the program is fairly straightforward.

Here are definitions of the variables:

ST$(49,1)	States/capitals array
K	Number of elements moved to the top of the list
R1%	State pointer
R2%	State or capital selector
AN$	Answer
RT%	Number right
WR%	Number wrong
HE%	Number of helps
I$	Temporary string for exchanging data

About Arrays

An *array* is simply an ordered set of data. It may have one or more *dimensions*. A one-dimensional array is merely a list whose data elements are numbered starting with 0. For example, a grocery list of 20 items, numbered from 0 to 19, would be a one-dimensional array with 20 data elements.

To define an array, you must use a special type of variable called a *subscripted variable*. This takes the form AN(I), where AN is the Array Name and I is the number (*subscript*) of the desired element. In our grocery list example, if I=19, then AN(I) would be the last item on the list. The first item on the list would be AN(0).

The array name may be any legal variable name, with $ (string variable) or % (integer variable) appended if appropriate. (This would indicate that the data elements contained in the array are strings or integers.)

Let's say you want a one-dimensional array with four elements. The four elements are integers (whole numbers): 21, 23, 25, and 27. The array would be represented by AN%(I). That is to say, AN%(0)=21, AN%(1)=23, AN%(2)=25, and AN%(3)=27.

A *two-dimensional array* is also an ordered list, but one whose elements are each an ordered list themselves. It's easier to understand if you picture it as a chart. For example, a two-dimensional array might look like this:

	I=0	I=1	I=2	I=3
J=0	21	23	25	27
J=1	43	45	47	49
J=2	51	53	58	59

A proper name for this array could be AN% and its elements identified as AN%(I,J). If I=0 and J=0, then AN%(I,J) =21. If I=3 and J=2, then AN%(I,J)=59. The advantage of arrays is that they let you store lots of numbers or other data without using lots of variables, and you can access any data element with a simple mathematical calculation. But be careful: arrays also consume big chunks of memory.

Arrays can become very complicated. It's easy to picture one- and two-dimensional arrays, but how about arrays of

three or even four dimensions? Elements of three- and four-dimensional arrays are identified in the form AN%(I,J,K) and AN%(I,J,K,L), respectively. IBM's Microsoft BASIC allows up to 255 dimensions and up to 32767 elements per dimension.

Creating Arrays

Typically, arrays are created with nested FOR-NEXT loops, each containing a READ from a DATA statement or an INPUT from a storage device. Each FOR-NEXT level creates one ordered list. For example, the following program could be used to define the contents of the two-dimensional array shown above:

```
10 DIM AN%(3,2)
20 FOR I=0 TO 3
30 FOR J=0 TO 2
40 READ AN%(I,J)
50 NEXT J
60 NEXT I
70 DATA 21,43,51,23,45,53,25,47,58,27,49,59
```

The inner (or nested) FOR-NEXT loop (lines 30–50) creates the ordered list of elements in the J-dimension within each element of the I-dimension. Compare the above table to the DATA statement in line 70 to see how the array is set up.

The DIMension statement (line 10) is required to tell the computer how much memory to set aside for the array. You can skip this for one-dimensional arrays with up to 11 elements. But arrays with more than 11 elements and all multi-dimensional arrays require a DIMension statement somewhere in the program *before* the first reference of the array variable. But don't place the DIMension statement where it will be executed more than once, or you'll get an error.

Note that dimension sizes in a DIMension statement are one less than the number of elements in the dimension, because counting starts at zero. The number of dimensions and the number of elements in each dimension are limited only by the amount of memory available. And arrays eat up memory *very* fast.

Remember that an array can hold other types of data besides numbers. States & Capitals Tutor uses a two-dimensional string array, ST$(I,J), to store the 50 states and 50 capitals.

States and Capitals Tutor

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
IJ 1Ø DEF SEG=Ø:POKE 1Ø47,64
PF 2Ø DIM ST$(49,1)
GO 3Ø FOR I=Ø TO 49
LK 4Ø FOR J=Ø TO 1
KK 5Ø READ ST$(I,J)
IN 6Ø NEXT J
HF 7Ø NEXT I
FF 8Ø K=Ø:RT%=Ø:WR%=Ø:HE%=Ø:MS=1
EI 9Ø COLOR 7:WIDTH 4Ø:KEY OFF:CLS:LOCATE 6,7,Ø:P
      RINT"STATES & CAPITALS TUTOR"
JM 1ØØ LOCATE 14:PRINT"This program tutors studen
       ts in"
AF 11Ø LOCATE ,3:PRINT"U.S. states and capitals."
NL 12Ø LOCATE 17:PRINT"If you don't know an answe
       r,"
MH 13Ø LOCATE ,3:PRINT"type HELP."
BP 14Ø WHILE RT%<5Ø:LOCATE 23,6,Ø:PRINT"<Press an
       y key to continue>";
JP 145 GOSUB 611
HD 15Ø A$="":WHILE A$="":A$=INKEY$:9=RND:WEND
FN 16Ø R1%=INT(RND*(5Ø-K))
GD 17Ø R2%=INT(2*RND)
GF 18Ø GOSUB 41Ø
OG 19Ø LOCATE 2,5,1,Ø,7
FK 2ØØ IF R2%=1 THEN PRINT "The capital of " ST$(
       R1%,Ø) " is?"; ELSE PRINT ST$(R1%,1) " is
       the capital of?";
KI 2Ø5 LOCATE 4,7
QN 2Ø8 A$=INKEY$:IF A$<>"" THEN 2Ø8
EI 21Ø INPUT"",AN$
CI 223 IF AN$="+" THEN MS=1:GOTO 21Ø
CA 225 IF AN$="-" THEN MS=Ø:GOTO 21Ø
JI 227 IF R1%=22 AND R2%=1 AND AN$="ST. PAUL" THE
       N GOSUB 26Ø:GOTO 23Ø
LC 228 IF AN$=ST$(R1%,R2%) THEN GOSUB 26Ø ELSE IF
        AN$="HELP" THEN GOSUB 35Ø ELSE GOSUB 38Ø
JM 23Ø GOSUB 45Ø
EN 24Ø WEND
DP 25Ø GOTO 5ØØ
HN 26Ø RT%=RT%+1
JC 27Ø LOCATE 8,14:PRINT"THAT'S RIGHT!"
CA 275 IF MS THEN SOUND 4ØØ,6:SOUND 5ØØ,2:SOUND 6
       ØØ,3.8:SOUND 32767,.2:SOUND 6ØØ,4:SOUND 4Ø
       Ø,8
DB 28Ø FOR I=Ø TO 1
LF 29Ø I$=ST$((49-K),I)
AA 3ØØ ST$((49-K),I)=ST$(R1%,I)
```

```
GF 310 ST$(R1%,I)=I$
NN 320 NEXT I
LM 330 K=K+1
MF 340 RETURN
PA 350 HE%=HE%+1
EA 360 LOCATE 8,4:PRINT"The answer is " ST$(R1%,R
       2%)
NL 370 RETURN
KI 380 WR%=WR%+1
NN 390 LOCATE 8,5:PRINT"No, the correct answer is
        "
NF 392 LOCATE 10,7:PRINT ST$(R1%,R2%)
IE 395 IF MS THEN SOUND 225,9:SOUND 150,10
MO 400 RETURN
FO 410 CLS:LOCATE 17,5:PRINT "Your score is:";
KD 420 PRINT SPC(9) " RIGHT"
BJ 430 PRINT SPC(27) " WRONG"
NJ 440 PRINT SPC(27) " HELPS"
JE 450 LOCATE 17,24
CG 460 PRINT RT%
LG 470 PRINT SPC(23) WR%
JH 480 PRINT SPC(23) HE%
NA 490 RETURN
GC 500 IF WR%+HE%=0 THEN 560
EH 510 PRINT"That's all, but not all your answers
             were correct, or I had to HELP you."
PM 520 PRINT "Want to try again (Y OR N)?";
JP 525 GOSUB 611
EG 530 A$="":WHILE A$="":A$=INKEY$:WEND
DP 540 IF A$<>"Y" AND A$<>"y" THEN END
BA 550 GOTO 80
NG 560 PRINT:PRINT"YOU DID IT!!!"
HB 570 PRINT"A perfect score and I didn't help!"
LO 580 PRINT "Do you want to try again (Y OR N)?"
       ;
FC 590 A$="":WHILE A$="":A$=INKEY$:WEND
DI 600 IF A$<>"Y" AND A$<>"y" THEN END
BJ 610 GOTO 80
JE 611 IF MS THEN SOUND 400,.3
NI 612 RETURN
NM 620 DATA ALABAMA,MONTGOMERY,ALASKA,JUNEAU,ARIZ
       ONA,PHOENIX,ARKANSAS,LITTLE ROCK
PE 630 DATA CALIFORNIA,SACRAMENTO,COLORADO,DENVER
       ,CONNECTICUT,HARTFORD,DELAWARE,DOVER
OF 640 DATA FLORIDA,TALLAHASSEE,GEORGIA,ATLANTA,H
       AWAII,HONOLULU,IDAHO,BOISE
AF 650 DATA ILLINOIS,SPRINGFIELD,INDIANA,INDIANAP
       OLIS,IOWA,DES MOINES,KANSAS,TOPEKA
PD 660 DATA KENTUCKY,FRANKFORT,LOUISIANA,BATON RO
       UGE,MAINE,AUGUSTA,MARYLAND,ANNAPOLIS
```

```
NF 670 DATA MASSACHUSETTS,BOSTON,MICHIGAN,LANSING
       ,MINNESOTA,SAINT PAUL,MISSISSIPPI,JACKSON
DO 680 DATA MISSOURI,JEFFERSON CITY,MONTANA,HELEN
       A,NEBRASKA,LINCOLN,NEVADA,CARSON CITY
LA 690 DATA NEW HAMPSHIRE,CONCORD,NEW JERSEY,TREN
       TON,NEW MEXICO,SANTA FE,NEW YORK,ALBANY
NI 700 DATA NORTH CAROLINA,RALEIGH,NORTH DAKOTA,B
       ISMARCK,OHIO,COLUMBUS
MA 710 DATA OKLAHOMA,OKLAHOMA CITY,OREGON,SALEM,P
       ENNSYLVANIA,HARRISBURG
EO 720 DATA RHODE ISLAND,PROVIDENCE,SOUTH CAROLIN
       A,COLUMBIA,SOUTH DAKOTA,PIERRE
OE 730 DATA TENNESSEE,NASHVILLE,TEXAS,AUSTIN,UTAH
       ,SALT LAKE CITY,VERMONT,MONTPELIER
GN 740 DATA VIRGINIA,RICHMOND,WASHINGTON,OLYMPIA,
       WEST VIRGINIA,CHARLESTON,WISCONSIN,MADISON
IN 750 DATA WYOMING,CHEYENNE
```

Word Hunt

Robert W. Baker
Translation by Patrick Parrish

Can you find the hidden words in this puzzle? Your score will be based on how quickly you locate them. It's not as easy as it seems.

There are five skill levels in this game, and you start off by selecting which level, from 1 to 5, you want to try. The easiest is level 1: You'll have one and a half minutes to locate each word. The toughest, level 5, allows you only about 20 seconds per word.

After deciding how much of a challenge you want, you enter ten words to prepare the game. If two people are playing, each one should privately enter the words for the other to find. The words can be from three to eight characters in length and contain only the letters *A–Z*. It's best to vary the word length since it will delay the game if there are too many long words. Should the computer be unable to fit the words into a puzzle, you'll be asked for a new list.

When you've provided the hidden words, hit any key and the game will start. You can score a maximum of 100, a minimum of 10 points per word. For example, if you get the correct word right away, you'll get the full 100 points, but the longer it takes you, the lower the point value. A wrong answer scores no points, and there's only one try per word.

You answer by typing in the row and column numbers of the first character of the word, followed by the direction code. Take a look at the diagram in the game to see how this works.

Major Variables

For those of you who are interested in game programming, we'll take a look at how the game is constructed. Many games follow this general pattern. But first, here is a list of the main variables in the program:

S	Defines the size of the letter matrix to be created.
W	Defines the number of words to be entered and used in the matrix.
M(S,S)	The actual letter matrix. Note that a floating-point numeric matrix is used instead of a string matrix. More about this later.
W$(W)	The word list.
L(W,3)	The starting location and direction of each word after it has been placed in the letter matrix. Each entry directly corresponds to the entry in the same position in the word matrix.
P(S,S) and **F(8)**	The working matrices that create the actual letter matrix used in the game.

Now, let's take a look at how the program itself works. First, the program gets the desired skill level (SL) as a number between 1 and 5. Lines 130–290 then get the list of words and check that each is a valid character string (A–Z). The words are put into the word list in alphabetic order as each word is entered by the user. This avoids the time-consuming process of sorting the entire word list at the end. In this way, there is a short delay as each word is entered, but it's not noticeable.

Line 340 initializes the letter matrix to all *'s (decimal value 42). Now each word in the word list is inserted randomly in the letter matrix in the following fashion:

1. The point matrix is cleared (line 360) so that we can remember what points in the matrix have been tried for a particular word in the word list.
2. Lines 400–440 check that there is still at least one point in the letter matrix that has not been tried (entry in P is still 0). If all points have been tried, the user is asked to enter a new list of words since this list will not fit properly in the letter matrix.
3. A random starting point (that has not been tried) is chosen in line 450.
4. The starting point is flagged as having been tried (P value now 1), and then a check is made to see if the matrix position is open (still *) or matches the first letter of the word (lines 460–470).

5. Now the direction matrix (F) is cleared to remember what directions have been tried from this starting point (line 490).
6. A check is made that at least one direction still hasn't been tried from this point (lines 500–510).
7. A random direction (that has not been tried) is chosen in line 520.
8. Then the word is checked to see if it can physically fit in the matrix in the selected direction from the current starting point (lines 530–650). This insures that the word will not exceed the boundaries of the letter matrix from this point.
9. If the word can fit, each character position in the selected direction is checked against the corresponding character of the word (lines 660–690). Each character in the matrix must match the corresponding character in the word or must be unused (still *).
10. If the word can be entered at this starting point and in this direction, each letter is inserted in the latter matrix (lines 710–720). Then the starting location and direction are saved for later use (line 740).
11. If the word will not fit, the next direction is tried until all directions are exhausted from this point.

When all words have been put into the matrix, the remaining unused positions (still *) are filled in with random letters (lines 760–770).

Everything is now set up to play the game as soon as the player hits a key (lines 780–800). The letter matrix is displayed along with a direction code diagram and a score box (lines 820–960). A word is given for the player to find in the matrix, and the timer is restarted (lines 970–1000). Then the program prompts the player for the starting location and direction code (lines 1020–1170). The values entered are then checked to see if they're correct, first against the values saved when the word was put into the matrix (lines 1190–1210). If the value does not match, the program checks to see if a "double" was created when the unused positions were filled with random letters. Thus, the program checks the player's answer again to insure that it is either right or wrong (lines 1230–1280). If an incorrect answer is entered, it is indicated, and the correct answer is displayed with no score added (lines 1360–1430). A

correct answer is indicated and the appropriate score displayed and added to the player's total. The score is based on the selected skill level and the time it takes to enter the answer.

A numeric vector was used for the actual letter matrix since it was easier and faster to use. Most people who have tried this game have found it interesting and fun to play. At times it can even be educational.

Word Hunt

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
EF 1Ø SCREEN Ø:WIDTH 4Ø:KEY OFF:COLOR 7,Ø,7:CLS
NN 2Ø LOCATE 11,16:PRINT"WORD HUNT":LOCATE 13,9:P
      RINT"(Caps Lock must be down)"
HH 3Ø LOCATE 22,1Ø:PRINT "Press any key to begin"
DL 4Ø A$=INKEY$:IF A$="" THEN A=RND(1):GOTO 4Ø
IC 8Ø S=1Ø:W=1Ø:DIM M(S,S),W$(W),P(S,S),L(W,3),F(
      8)
QK 9Ø CLS:PRINT:PRINT:PRINT "What skill level ?"
DP 1ØØ PRINT:PRINT "1 (Easy) to 5 (Hard)    3"STRI
       NG$(3,29);:INPUT R$:IF R$="" THEN R$="3"
DB 11Ø X=VAL(R$):IF X<1 OR X>5 THEN 9Ø
HM 12Ø SL=6-X
AE 13Ø PRINT:PRINT:PRINT "Enter"W"words."
BF 14Ø PRINT "Make each word 3 to 8 characters lo
       ng.":PRINT
MF 15Ø REM *** GET WORDS & PUT IN ORDER ***
HA 16Ø REM *** LONGEST TO SHORTEST ***
LL 17Ø FOR X=1 TO W:L(X,1)=Ø:L(X,2)=Ø:L(X,3)=Ø
QI 18Ø PRINT "Word"X"    "STRING$(3,29);
NE 19Ø INPUT R$:Q=LEN(R$)
EN 2ØØ IF Q<3 THEN PRINT TAB(24)STRING$(1,3Ø)"* T
       OO SHORT *":GOTO 18Ø
GF 21Ø IF Q>8 THEN PRINT TAB(24);STRING$(1,3Ø);"*
        TOO LONG *":GOTO 18Ø
GE 22Ø X9=Ø:FOR Y=1 TO Q:A=ASC(MID$("*"+R$+"*",Y+
       1,1))
MG 23Ø IF A<65 OR A>9Ø THEN X9=1:Y=Q
GA 24Ø NEXT:IF X9=1 THEN PRINT TAB(24)STRING$(1,3
       Ø)"* BAD WORD *":GOTO 18Ø
DH 25Ø IF X=1 THEN W$(X)=R$+"*":GOTO 29Ø
BF 26Ø X9=Ø:FOR Y=1 TO X-1:IF Q<=LEN(W$(Y))-1 THE
       N 28Ø
QO 27Ø FOR B=X TO Y+1 STEP -1:W$(B)=W$(B-1):NEXT
       B:W$(Y)=R$+"*":X9=1:Y=X-1
HK 28Ø NEXT Y:IF X9=Ø THEN W$(X)=R$+"*"
HA 29Ø NEXT X
GM 3ØØ CLS:LOCATE 6,11:PRINT "That's enough words
       !"
```

```
OH 310 LOCATE 12,11:PRINT "Please be patient..."
NN 320 LOCATE 18,10:PRINT "I'm making the puzzle!
       "
BK 330 REM *** INITIALIZE LETTER MATRIX ***
AH 340 FOR X=1 TO S:FOR Y=1 TO S:M(Y,X)=42:NEXT Y
       :NEXT X:Q=0
EI 350 REM *** INIT POINT MATRIX & GET NEXT WORD
       ***
HG 360 FOR X=1 TO S:FOR Y=1 TO S:P(Y,X)=0:NEXT Y
EK 370 NEXT X:Q=Q+1:IF Q>W THEN 760
HJ 380 G=LEN(W$(Q))-2
IO 390 REM *** TRY ALL POINTS FOR EACH WORD ***
FI 400 X9=0:FOR X=1 TO S:FOR Y=1 TO S:IF P(Y,X)=0
        THEN X9=1:X=S:Y=S
PE 410 NEXT Y:NEXT X:IF X9=1 THEN 450
GA 420 REM *** WORD WILL NOT FIT, TRY AGAIN! ***
KG 430 CLS:PRINT "This list of words will not all
        fit."
MM 440 PRINT:PRINT "Please enter new words.":GOTO
        130
BN 450 A=INT(S*RND(1)+1):B=INT(S*RND(1)+1):IF P(B
       ,A)<>0 THEN 450
KL 460 P(B,A)=1:IF M(B,A)=42 THEN 490
NA 470 IF M(B,A)<>ASC(LEFT$(W$(Q),1)) THEN 400
GA 480 REM *** TRY ALL DIRECTIONS FROM THIS POINT
        ***
EI 490 FOR X=1 TO 8:F(X)=0:NEXT X
OG 500 X9=0:FOR X=1 TO 8:IF F(X)=0 THEN X9=1:X=8
JM 510 NEXT X:IF X9=0 THEN 400
FN 520 D=INT(8*RND(1)+1):IF F(D)=1 THEN 520
DB 530 F(D)=1:ON D GOTO 550,590,580,620,610,650,6
       40,560
FF 550 IF (A+G)>S THEN 500
IO 560 IF (B-G)<1 THEN 500
JN 570 GOTO 670
GE 580 IF (B+G)>S THEN 500
GN 590 IF (A+G)>S THEN 500
IA 600 GOTO 670
HM 610 IF (A-G)<1 THEN 500
GJ 620 IF (B+G)>S THEN 500
IG 630 GOTO 670
IL 640 IF (B-G)<1 THEN 500
HE 650 IF (A-G)<1 THEN 500
HD 660 REM *** CHECK WORD MATCHES INTO MATRIX ***
BL 670 X=A:Y=B:X9=0:FOR N=2 TO G+1:GOSUB 1550:IF
       M(Y,X)=42 THEN 690
JD 680 IF M(Y,X)<>ASC(MID$(W$(Q),N,1)) THEN X9=1:
       N=G+1
LA 690 NEXT N:X=A:Y=B:IF X9=1 THEN 500
IK 700 REM *** ENTER WORD ***
```

```
HF 71Ø FOR N=1 TO G+1:IF M(Y,X)=42 THEN M(Y,X)=AS
       C(MID$(W$(Q),N,1))
IM 72Ø GOSUB 155Ø:NEXT N
HJ 74Ø L(Q,1)=A-1:L(Q,2)=B-1:L(Q,3)=D:IF Q<W THEN
        36Ø
GL 75Ø REM *** FILL IN SPACES ***
KO 76Ø FOR Y=1 TO S:FOR X=1 TO S:IF M(Y,X)=42 THE
       N M(Y,X)=INT(25*RND(1)+65)
CK 77Ø NEXT X:NEXT Y:WP=Ø:TS=Ø
HP 78Ø CLS:LOCATE 1Ø,1:PRINT "READY !"
CC 79Ø LOCATE 16,9:PRINT "Press any key to play !
       "
OJ 8ØØ R$=INKEY$:IF R$="" THEN 8ØØ
MP 81Ø REM *** SET UP DISPLAY ***
HJ 82Ø CLS:PRINT:PRINT "        COLUMN"TAB(25)"W O
       R D"
OB 83Ø REM *** PRINT 'ROW' DOWN LEFT COLUMN ***
EA 86Ø PRINT STRING$(5,31)"R"STRING$(1,31)STRING$
       (1,29)"O"STRING$(1,31)STRING$(1,29)"W"STRI
       NG$(7,3Ø)STRING$(3,28);
GP 87Ø FOR X=Ø TO S-1:PRINT RIGHT$(STR$(X),1);:NE
       XT X:PRINT:Y=1:GOSUB 165Ø
AC 88Ø FOR Y=1 TO S:PRINT STRING$(2,28)RIGHT$(STR
       $(Y-1),1)STRING$(1,179);
JD 89Ø FOR X=1 TO S:PRINT CHR$(M(Y,X));:NEXT X
LD 9ØØ PRINT STRING$(1,179):NEXT Y:Y=Ø:GOSUB 165Ø
OI 91Ø PRINT:PRINT"   DIRECTIONS:":COLOR Ø,7:PRIN
       T STRING$(1,31)STRING$(6,28)"7 8 1"
GC 92Ø PRINT STRING$(6,28)" \"STRING$(1,179)"/ ":
       PRINT STRING$(6,28)"6"STRING$(1,196)STRING
       $(1,4)STRING$(1,196)"2"
JE 921 PRINT STRING$(6,28)" /"STRING$(1,179)"\ ":
       PRINT STRING$(6,28)"5 4 3":COLOR 7,Ø
GD 93Ø G=13:GOSUB 17ØØ:PRINT STRING$(27,28)"SCORE
       ":PRINT:PRINT STRING$(25,28)STRING$(1,218)
       STRING$(7,196)STRING$(1,191)
II 935 PRINT STRING$(25,28)STRING$(1,179)"
       "STRING$(1,179)
BL 94Ø PRINT STRING$(25,28)STRING$(1,179)"  Ø
       "STRING$(1,179)
AB 95Ø PRINT STRING$(25,28)STRING$(1,179)"
       "STRING$(1,179):PRINT STRING$(25,28)STRING
       $(1,192)STRING$(7,196)STRING$(1,217);
QO 96Ø G=Ø:GOSUB 17ØØ:PRINT "                   "
       :REM 19 spaces
OB 97Ø WP=WP+1:IF WP>W THEN 145Ø
ED 98Ø Q=LEN(W$(WP))-1
EF 99Ø REM *** NEXT WORD ***
NK 1ØØØ GOSUB 17ØØ:PRINT TAB(28-(Q/2))LEFT$(W$(WP
        ),Q)
```

```
CE 1ØØ5 TI=Ø
NE 1Ø2Ø G=3:GOSUB 17ØØ:PRINT "STARTING LOCATION"
DK 1Ø25 PRINT STRING$(19,28)"(ROW,COLUMN):"
GH 1Ø3Ø FOR G=5 TO 12:GOSUB 17ØØ
MI 1Ø4Ø PRINT "                    ":NEXT G:G=5:G
        OSUB 17ØØ:REM 2Ø SPACES
BB 1Ø5Ø B$=INKEY$:IF B$="" THEN TI=TI+1:GOTO 1Ø5Ø
CD 1Ø6Ø IF ASC(B$)=13 THEN 1Ø5Ø
HO 1Ø7Ø PRINT B$",";:IF B$="Ø" THEN B=Ø:GOTO 1Ø9Ø
AH 1Ø8Ø B=VAL(B$):IF B<1 OR B>9 THEN PRINT STRING
        $(2,29)"  "STRING$(2,29);:GOTO 1Ø5Ø
MI 1Ø9Ø A$=INKEY$:IF A$="" THEN TI=TI+1:GOTO 1Ø9Ø
HC 11ØØ IF ASC(A$)=13 THEN 1Ø9Ø
BG 111Ø PRINT A$:IF A$="Ø" THEN A=Ø:GOTO 114Ø
QO 112Ø A=VAL(A$):IF A<1 OR A>9 THEN 1Ø3Ø
PN 113Ø REM *** GET DIRECTION ***
MC 114Ø G=7:GOSUB 17ØØ:PRINT "DIRECTION:":PRINT S
        TRING$(19,28)" "STRING$(1,29);
HB 115Ø D$=INKEY$:IF D$="" THEN TI=TI+1:GOTO 115Ø
GJ 116Ø IF ASC(D$)=13 THEN 115Ø
LC 117Ø PRINT D$:D=VAL(D$):IF D<1 OR D>8 THEN 114
        Ø
KI 118Ø REM *** CHK IF GOOD INFO INPUT ***
OH 119Ø WT=INT(TI/14Ø)
PK 1195 IF B<>L(WP,2) THEN 123Ø
KD 12ØØ IF A<>L(WP,1) THEN 123Ø
NC 121Ø IF D=L(WP,3) THEN 136Ø
GN 122Ø REM *** CHK IF A DOUBLE MAY EXIST ***
CG 123Ø X=A+1:Y=B+1:G=LEN(W$(WP))-1:IF M(Y,X)<>AS
        C(LEFT$(W$(WP),1)) THEN 13ØØ
OH 124Ø X9=Ø:FOR N=2 TO G:GOSUB 155Ø:IF X<1 OR X>
        1Ø THEN 127Ø
EC 125Ø IF Y<1 OR Y>1Ø THEN 127Ø
AB 126Ø IF M(Y,X)=ASC(MID$(W$(WP),N,1)) THEN 128Ø
PP 127Ø X9=1:N=G
NA 128Ø NEXT N:IF X9=Ø THEN 136Ø
PK 129Ø REM *** BAD START/DIR - NO SCORE ***
IP 13ØØ G=5:GOSUB 17ØØ:B$=STR$(L(WP,2)):A$=STR$(L
        (WP,1))
KO 131Ø PRINT RIGHT$(B$,LEN(B$)-1)","RIGHT$(A$,LE
        N(A$)-1)
HK 132Ø G=8:GOSUB 17ØØ:PRINT STRING$(1,29)L(WP,3)
IO 133Ø G=1Ø:GOSUB 17ØØ:PRINT "^"SPC(13)"^"
FD 134Ø G=11:GOSUB 17ØØ:PRINT STRING$(1,192)" No,
         Correct "STRING$(1,217):GOTO 142Ø
QL 135Ø REM *** GOOD ANSWER - GET  SCORE ***
FC 136Ø IF WT<(SL*5) THEN WS=1ØØ:GOTO 139Ø:REM MA
        XIMUM SCORE
HF 137Ø IF WT>(SL*2Ø) THEN WS=1Ø:GOTO 139Ø:REM MI
        NIMUM SCORE
```

```
OJ 1380 WS=10+(SL*20-WT)
BE 1390 G=10:GOSUB 1700:PRINT"^"
AB 1400 G=11:GOSUB 1700:PRINT STRING$(1,192)" Yes
        ,"WS"Points":TS=TS+WS
BE 1410 REM *** UPDATE TOTAL SCORE ***
NN 1420 G=18:GOSUB 1700:PRINT STRING$(8,28)TS
EL 1430 FOR X=1 TO 1750:NEXT X:GOTO 960
MM 1440 REM *** END GAME ***
GF 1450 PRINT STRING$(1,11):REM STRING$(15,31)
BM 1460 FOR X=1 TO 16:PRINT "
                                ":NEXT X:REM 39 spaces
OF 1470 FOR X=1 TO 6
HG 1480 PRINT "                 ":NEXT X:REM 17 s
        paces
EK 1490 PRINT STRING$(1,11)STRING$(20,31)"Play Ag
        ain (Y or N) ?"
PD 1500 R$=INKEY$:IF R$="" THEN 1500
IN 1510 IF R$="y" OR R$="Y" THEN 90
BH 1515 IF R$<>"n" AND R$<>"N" THEN 1500
JI 1525 CLS:PRINT TAB(5)"Thank you for playing WO
        RD HUNT.
OP 1526 PRINT :PRINT TAB(5)"Hope you had fun."
FC 1530 PRINT :PRINT TAB(5)"See you later!!!"
OE 1535 FOR X=1 TO 5000:NEXT X:CLS:END
OC 1540 REM *** SUBR TO INCREMENT COORDINATES IN
        DIR ***
PP 1550 ON D GOTO 1560,1570,1580,1590,1600,1610,1
        620,1630
KF 1560 Y=Y-1
EA 1570 X=X+1:RETURN
JL 1580 X=X+1
FE 1590 Y=Y+1:RETURN
ID 1600 Y=Y+1
EC 1610 X=X-1:RETURN
JN 1620 X=X-1
FG 1630 Y=Y-1:RETURN
LG 1640 REM *** SUBR FOR BOX TOP/BOTTOM ***
QI 1650 PRINT STRING$(3,28);:IF Y=1 THEN PRINT ST
        RING$(1,218);:GOTO 1670
OB 1660 PRINT STRING$(1,192);
FP 1670 FOR X=0 TO S-1:PRINT STRING$(1,196);:NEXT
         X:IF Y=1 THEN PRINT STRING$(1,191):RETUR
        N
QN 1680 PRINT STRING$(1,217):RETURN
IP 1690 REM *** SUBR TO POSITION ***
LN 1700 PRINT STRING$(1,11)STRING$(19,28);:FOR X9
        =1 TO G+3:PRINT STRING$(1,31);:NEXT X9:RE
        TURN
```

Spelling Bee

Daniel Bonachea
Translation by Gregg Peele

"Spelling Bee" is an educational spelling game for both the IBM PC and PCjr. The program works in all versions of IBM BASIC. By choosing one of three learning levels, you can target the game for children of various ages.

Based on the flashcard method of teaching spelling, "Spelling Bee" is an educational and entertaining game for children of different learning levels. The game uses a small number of the more important keys, so a child will also become familiar with their functions.

How to Use Spelling Bee

The computer first asks for the child's name. This is followed by a yes/no option for game instructions. Spelling Bee requires use of the left and right cursor control keys, Delete, Insert, and the cursor-down key.

Before the game begins, you must also choose one of three levels: Easy, Medium, or Hard. The words range in difficulty from *cat* or *ant* (Easy), *snake* or *couch* (Medium), to *stereo* or *committee* (Hard).

The game begins by displaying the alphabet with a pointer under the letter *M*. Score and word number are displayed near the top. Each game begins with a perfect score (75). If the child can maintain this score, he or she has played a perfect game. A word is flashed near the top of the screen, and the child begins to spell by moving the line with the left or right arrow key to a letter of the alphabet. Pressing the Insert key prints that letter near the top of the screen. Should the child decide that the letter is incorrect, the Delete key will erase it.

When the child is satisfied with the word's spelling, a press of the down-arrow key prompts the computer to respond with "You spelled it correctly" or "That's incorrect. Try again." The child has three chances to spell a word correctly. The computer will spell the word if all three chances are used.

It then goes on to the next word. A session can be ended by typing XX. An option to play again at any of the levels then follows.

Program Modifications

You can make several simple program modifications to tailor the program for any child's needs. The DATA statements containing the spelling words begin at lines 450 (Easy), 470 (Medium), and 500 (Hard). You can insert any words of your choice at these line numbers as long as there are 75 total words and 25 in each of the three levels. You'll want to do this once your child has played the game a number of times and has mastered the existing words.

If you wish to increase or decrease the length of time the spelling word is flashed on the screen, change the value (1000) in the FOR-NEXT loop in line 620. A larger value increases the time the word is displayed, and vice versa.

Spelling Bee

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KJ 10 DEF SEG = &HB800
BF 20 REM Spelling Bee IBM PC & PCjr
CC 30 DIM WORD$(75)
CJ 40 KEY OFF
DE 50 FOR T= 1 TO 75 :READ WORD$(T):NEXT
CL 60 WIDTH 40
MH 70 DIM NA$(20):DIM L$(75):SCREEN 0:CLS:LOCATE
      3,8:PRINT "S P E L L I N G  B E E"
PN 80 LOCATE 6,4:PRINT"Hello, my name is Spelling
      Bee."
CL 90 ON ERROR GOTO 260
MN 100 LOCATE 11,4:INPUT"What is your name";NA$
OD 110 IF NA$="" THEN PRINT :PRINT "Don't you hav
       e a name?":GOTO 90
NJ 120 IF ASC(NA$)>96 THEN X=ASC(NA$):X=X-32:MID$
       (NA$,1,1)=CHR$(X)
KM 130 LOCATE 16,4:PRINT"Hello, ";NA$;".":LOCATE
       21,4:PRINT"Would you like to see instructi
       ons":LOCATE 22,4:PRINT"(Y or N)?"
EO 140 A$=INKEY$:A$=LEFT$(A$,1):IF A$="y" OR A$="
       Y" THEN 290
FK 150 IF A$="n" OR A$="N" THEN 180
DL 160 IF A$="" THEN 140
CK 170 PRINT "Y or N only, please.":GOTO 140
GC 180 REM Choose Difficulty
BP 190 CLS
```

```
AG 200 LOCATE 6,6:PRINT"How hard do you want your
        words, ":LOCATE 7,6:PRINT NA$" ?":LOCATE
       11,6:PRINT"Press a number from 1 to 3:"
JP 210 LOCATE 14,15:PRINT"1) Easy":LOCATE 16,15:P
       RINT"2) Medium":LOCATE 18,15:PRINT"3) Hard
       "
OO 220 A$=INKEY$:IF A$="" THEN 220
LI 230 IF VAL(A$)<1 OR VAL(A$) >3 THEN 250
HP 240 GOTO 560
KN 250 CLS:PRINT"You pressed the wrong key. Pleas
       e try   again, ";NA$;".":PRINT:GOTO 200
LN 260 REM Name Entry Error Trap
GD 270 PRINT "You made a mistake.  Please try aga
       in."
CN 280 GOTO 90
HP 290 REM Instructions
IB 300 CLS:LOCATE 3,1:PRINT"Great!  Now, ";NA$;",
       ":PRINT "all you have to do is follow thes
       e       simple directions:"
IE 310 LOCATE 9,1:PRINT"A word will be flashed br
       iefly on the   screen.  After the word dis
       appears,      spell it by moving the arrow
       under       the letter and hitting the 'Ins
       ' key.
JA 320 LOCATE 16,1:PRINT"If you accidentally type
        the wrong       letter, you can erase it w
       ith the        'Del' key.":PRINT
II 330 LOCATE 22,7:PRINT"Press any key to continu
       e"
EM 340 A$=INKEY$:IF A$="" THEN 340
NO 350 CLS:PRINT:PRINT"When you've spelled the wo
       rd, press the down arrow key ("CHR$(25)")
LN 360 LOCATE 6,1:PRINT"You get three chances to
       spell each wordcorrectly.  If you cannot s
       pell the       word after three tries, I wil
       l show you the correct spelling."
ED 370 LOCATE 12,1:PRINT"You start with a score o
       f 75 points.     Each time you misspell a w
       ord, you lose 1 point.  Easy, huh?"
HO 380 PRINT:PRINT:PRINT"If you wish to end a ses
       sion, simply"
OL 390 PRINT"spell 'XX' instead of the "
IK 400 PRINT"word you would normally spell.":PRIN
       T
HF 410 LOCATE 22,7:PRINT"Press any key to continu
       e"
CK 420 A$=INKEY$:IF A$="" THEN 420
GN 430 GOTO 180
PB 440 REM words
```

```
QM 450 DATA CAT,DOG,ANT,AND,ANY.AN,AM,CAN,CAP,TOP
       ,STOP,POT,TAP,PAT,CAR,CART,ART
NH 460 DATA HAND,HAT,FOOT,BOOK,FLY,SKY,SAW,SEE
NH 470 DATA SNAKE,SNACK,BOAT,MANY,LOOSE,LOSE,CHOO
       SE,CHOSE,CHASE,CHEESE,STOVE,STORE
GC 480 DATA STEAL,STAIRS,WHOLE,SCREW,WASHER,HORSE
       ,STEER,STONE,PLANT,RADIO,COUCH
NI 490 DATA CHAIR,TABLE
KC 500 DATA STEREO,STATION,TELEVISION,CUSHION,CAU
       TION,FREEZER,WEATHER,WHETHER
GF 510 DATA WHOEVER,HAMMOCK,COMMITTEE,COMPUTER,LI
       CENSE,MONITOR,DICTIONARY,RECEIVE
AE 520 DATA RECORD,SPEAKER,CURTAIN,PILLOW,WATERBE
       D,WINDOW,THEATER,PIANO,LIVER
OP 530 REM lets go
PE 540 PRINT "Press any key to go"
NP 550 A$=INKEY$:IF A$=""THEN 550
CM 560 LEV = VAL(A$)
EF 570 Z= 1+25*(LEV-1): TRY = 3:SC =75
BB 580 CLS
HO 590 IF SC = 0 THEN 910 ELSE LOCATE 2,16:PRINT
       "SCORE ";SC
PE 600 LOCATE 3,16:PRINT "Word #";Z-25*(LEV-1)
GP 610 LOCATE 6,(20-.5*LEN(WORD$(Z))):PRINT WORD$
       (Z)
BJ 620 FOR T= 1 TO 1000:NEXT
BL 630 LOCATE 6,8:PRINT STRING$(20,32)
PH 640 REM set up alphabet
FK 650 FOR T= 1 TO 3000 :NEXT
FI 660 R= 65:FOR T= 1054 TO 1054+50 STEP 2: POKE
       T,R :R=R+1: NEXT
IJ 670 REM move arrow
LB 680 LC = 1158:POKE LC,24:POKE LC+1,15
AG 690 P=0
OP 700 LOCATE 9,20-.5*LEN(WORD$(Z)):D$= STRING$(L
       EN(WORD$(Z)),95):PRINT D$
PM 710 NI$="":IN$= INKEY$:IF IN$=""THEN 710
PC 720 IF LEN(IN$)=2 THEN NI$=RIGHT$(IN$,1)ELSE N
       I$= ""
JJ 730 IF NI$= CHR$(83)AND P>0 THEN MID$(D$,P)=CH
       R$(95):LOCATE 9,20-.5*LEN(WORD$(Z)):PRINT
       D$:P=P-1
HE 740 OC = LC:IF NI$=CHR$(77)THEN LC = LC+2 : GO
       TO 780
JM 750 LT$= CHR$(PEEK (LC-80))
EP 760 IF NI$=CHR$(82)AND P<LEN(WORD$(Z))THEN P=P
       +1:MID$(D$,P)=LT$:LOCATE 9,20-.5*LEN(WORD$
       (Z)):PRINT D$
NF 770 IF NI$=CHR$(75)THEN LC =LC-2
BA 780 IF LC > 1184 THEN LC = LC-52
```

```
HF 79Ø IF LC < 1134 THEN LC = LC+52
NL 8ØØ IF LC <> OC THEN POKE OC,32:POKE LC+1,15 :
       POKE LC,24
MO 81Ø IF NI$ = CHR$(8Ø) THEN GOSUB 85Ø :GOTO 58Ø
FO 82Ø GOTO 71Ø
CH 83Ø REM SPELL OUT WORD
GC 84Ø FOR T= 1 TO LEN (WORD$(Z)):LOCATE 11,2Ø-.5
       *LEN(WORD$(Z)):PRINT LEFT$(WORD$(Z),T):FOR
        K= 1 TO 1ØØØ:NEXT: NEXT:FOR T= 1 TO 1ØØØ
       :NEXT :Z=Z+1:RETURN
PN 85Ø IF LEFT$(D$,1)= "X" THEN 91Ø ELSE LOCATE 2
       Ø,8:IF D$= WORD$(Z) THEN PRINT"You spelled
        it correctly.":Z=Z+1:TRY = 3:GOSUB 88Ø EL
       SE PRINT "That's incorrect. Try again.":SC
        = SC-1:TRY=TRY-1:IF TRY<=Ø THEN GOSUB 83Ø
       :GOSUB 88Ø:TRY= 3:IF FLAG =1 THEN 91Ø
PF 86Ø FOR T= 1 TO 15ØØ:NEXT: LOCATE 2Ø,15:PRINT
       STRING$(26,32)
NA 87Ø RETURN
LP 88Ø FOR T= 1 TO 15ØØ:NEXT :LOCATE 2Ø,8:PRINT S
       TRING$(36,32): IF LEV < 3  AND Z=26 OR Z=5
       1 THEN LOCATE 2Ø,8:PRINT "     On to the ne
       xt level":LEV= LEV +1:FOR T = 1 TO 15ØØ :
       NEXT
II 89Ø IF Z >= 75 THEN LOCATE 2Ø,4:PRINT" You hav
       e spelled all the words.":Z=1:LEV = 1:FOR
       T= 1 TO 15ØØ:NEXT:FLAG=1
MD 9ØØ RETURN
IJ 91Ø CLS :LOCATE 8,16:PRINT"Score";SC
MG 92Ø LOCATE 12,11:PRINT"Play Again Y or N ?"
OM 93Ø A$=INKEY$:IF A$ ="" THEN 93Ø
NJ 94Ø IF A$="y"OR A$="Y"THEN RUN ELSE END
```

Gradebook

Stephen Levy
Translation by Gregg Peele

"Gradebook" is a valuable organizational tool for teachers. It handles student lists, grading conversions, grade averaging, and much more. Written for the IBM PC and PCjr with a disk drive.

Schools continue to buy computers for students to use and learn on. But why not let teachers take advantage of the convenience that a computer provides? "Gradebook" is a tool that is easy to use and replaces the periodic drudgery of averaging grades.

Gradebook will keep a record of students' grades and assignments for up to 45 students on one disk. In addition, the program will average grades and display grades or assignments to the screen, or list them to a printer. Grades may be either numbers or letters or a mixture of both. Letter grades can be entered with a plus or minus sign.

The program is easy to use. It has nine menu options, one of which allows you to correct a student's name or any grade. Another feature gives you the ability to transfer the names of all the students from one disk to another, without transferring the grades. Thus, you won't need to reenter each student's name at the beginning of each marking period or when a disk fills up.

Menu Options

When you first run the program, you'll see the menu screen. The nine menu options are

1. Read Grades
2. Read Assignments
3. Enter Names
4. Enter Grades
5. Enter Assignments
6. Print Grades or Assignments
7. Make Correction
8. Initialize a Disk
9. End

You may return to the menu any time by typing XXX when an input is asked for.

The first time you use the program, select option 3. If you pick another option, the program will tell you that there are no student grades on the disk. You will also get this error message if you happen to have the wrong disk in the drive.

Here is a discussion of each of the menu options.

Read Grades. This option produces a list of the last names of all students previously entered (option 3) onto this disk. You will be prompted for the *numbers* of the first and last students whose grades and averages you wish to see. However, on each screen display you are limited to viewing two to five students' grades at a time.

Read Assignments. With option 2 you'll get a list of previously entered (option 4) assignments on this disk.

Enter Names. This is where you will begin the program the first time it is used each school year. Enter Names lets you enter and add new students to the list of students. Remember that only 45 students are allowed on one disk—first name up to nine characters, last name up to ten characters, no middle names.

Enter Grades. This option is the heart of the program. It produces a list of students previously entered (option 3) and asks which student's grades you wish to enter. The program accepts any one-, two-, or three-digit number as well as the letters A, B, C, D, E, and F with or without a plus or minus. When grades are averaged, letter grades are converted to numbers as follows:

A+ = 97	A = 93	A− = 89
B+ = 87	B = 83	B− = 79
C+ = 77	C = 73	C− = 69
D+ = 67	D = 63	D− = 59
E+ = 54	E = 50	E− = 46
F+ = 54	F = 50	F− = 46

If you wish to change these conversions you must change the figures in two places. First, change the display on the screen, lines 990 to 1000. Second, change the actual conversions on lines 500 to 640 to suit your needs.

Enter Assignments. When you request this option, the Gradebook will list all previously entered assignments and allow you to add to the list. The assignment length must be no greater than 28 characters (including blank spaces). You

can also use this option for messages or notes. It functions like a notepad and has no real bearing on students' grades, averaging, and so forth.

Print Grades or Assignments. If you need a hardcopy of your students' grades, this is the option to use. It allows you to print all or some of the students' names, grades, and averages. It will also print a hardcopy of the list of assignments stored on the disk.

Make Correction. Option 7 permits you to change any student's name or grade.

Initialize a Disk. With this option, you'll save the time and effort of reentering students' names each marking period or when a disk gets full. The names of all the students stored on one disk will be transferred automatically to a new disk without transferring grades.

End. By using this option to exit the program, you assure the closing of all files. It is imperative that you *never end a session by just turning off the computer or disk drive. Always* use option 9.

Remember, typing XXX in response to any prompt will return you to the menu.

Suggestions for Easier Use

Entering multiple grades for one student is much faster than entering only one or two grades at a time. In other words, wait until you have five or six grades for each student before entering grades. You will soon discover that if it takes you 10 minutes to enter two grades for each of 30 students, it will take only 12 or 13 minutes to enter five grades for each student if done at one session.

It's good practice to check the amount of space left on a disk before you begin a session on a disk you've been using awhile. After using Gradebook you will soon learn approximately how many grades will fit on one disk.

Back up your data. Either back up your disk or maintain a written record just in case the disk gets damaged or lost.

Gradebook

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
BG 10 KEY OFF
KI 20 DIM NA$(50,2),GA$(100),GR$(100),AS$(100)
MM 30 WIDTH  40
HG 40 COLOR 7,0
```

```
AB 5Ø CLS:ON ERROR GOTO Ø
FB 6Ø LOCATE 1,11:PRINT "G R A D E B O O K"
CC 7Ø LOCATE 3,5:PRINT"YOUR OPTIONS ARE:"
PA 8Ø LOCATE 5,5: PRINT "1. Read Grades"
EM 9Ø LOCATE 7,5: PRINT"2. Read Assignments"
MM 1ØØ LOCATE  9,5 :PRINT "3. Enter Names"
IF 11Ø LOCATE 11,5 : PRINT "4. Enter Grades"
EN 12Ø LOCATE 13,5 : PRINT "5. Enter Assignments"
PF 13Ø LOCATE 15,5: PRINT "6. Print Grades or Ass
       ignments"
JI 14Ø LOCATE 17,5:PRINT "7. Make Corrections"
FE 15Ø LOCATE 19,5:PRINT "8. Initialize a Disk"
HE 16Ø LOCATE 21,5 : PRINT "9. End"
PK 17Ø LOCATE 23,5:PRINT "Enter your Choice"
GO 18Ø A$=INKEY$:IF VAL(A$)<1 OR VAL(A$)>9 THEN 1
       8Ø
LD 19Ø A= VAL(A$):IF A= 9 THEN END:REM close all
       files
MD 2ØØ CLS :COLOR 23,Ø :LOCATE 6,6:PRINT"BE SURE
       DATA DISK IS IN DRIVE A"
BN 21Ø FOR T = 1 TO 1ØØØ: NEXT
GK 22Ø COLOR 7,Ø
MO 23Ø LOCATE 1Ø,8: PRINT "You may enter XXX to a
       ny"
EH 24Ø LOCATE 12,8: PRINT "prompt to return to me
       nu."
LP 25Ø LOCATE 17,7:PRINT "PRESS THE SPACE BAR TO
       BEGIN"
LE 26Ø A$=INKEY$:IF A$<>" " THEN 26Ø
MG 27Ø ON ERROR GOTO Ø:CLOSE #1:IF A=2 OR A=5 OR
       A=8 THEN 29Ø
CM 28Ø IF A=2 OR A=5 OR A=8 THEN 29Ø
ØN 29Ø ON A GOTO 31Ø,66Ø,79Ø,95Ø,128Ø,146Ø,183Ø,2
       53Ø,3ØØ
LM 3ØØ END
PD 31Ø REM read students to put on screen
JF 32Ø IF R=Ø THEN GOSUB 268Ø:ELSE GOSUB 277Ø:ON
       ERROR GOTO Ø
CC 33Ø IF  NS = 1 THEN LOCATE 12,6:PRINT "No stud
       ents have been entered":FOR T = 1 TO 1ØØØ:
       NEXT :GOTO 5Ø
NM 34Ø LOCATE 23,1:INPUT "First student number ";
       FI$:FI=VAL(FI$):IF FI$="XXX" OR FI$="xxx"
       THEN 5Ø
CG 35Ø LOCATE 23,1:PRINT STRING$(25,32):LOCATE 23
       ,1 :INPUT"Last student number ";LA$:LA=VAL
       (LA$):IF LA$="XXX" OR LA$="xxx" THEN 5Ø
EA 36Ø IF LA<FI THEN CLS:GOSUB 284Ø:GOTO 31Ø
OE 37Ø IF LA>NS-1 THEN CLS:GOSUB 284Ø :GOTO 31Ø
```

```
MA 380 IF FI<>0 AND LA <>0 THEN 390 ELSE CLS:GOSU
       B 2840:GOTO 310
IC 390 CLS:FOR T= FI TO LA :CLOSE 1: FILE$ = "stu
       " +RIGHT$(STR$(T),LEN(STR$(T))-1):OPEN FIL
       E$ FOR INPUT AS  1
JJ 400 IF T= 10 OR T= 20 OR T =30 OR T=40 THEN GO
       SUB 2820 :CLS
HH 410 INPUT #1,NA$(T,1): INPUT #1,NA$(T,2):Z=1:
       REM get both names
JL 420 IF EOF(1)THEN GR$(1)= "No grades":GOTO 440
LB 430 INPUT #1,GR$(Z):IF EOF(1) THEN 440:ELSE Z=
       Z+1:GOTO 430
OE 440 GR= 0:CLOSE 1
QP 450 PRINT LEFT$(NA$(T,1),8);" ";LEFT$(NA$(T,2)
       ,8);" ";:FOR J= 1 TO Z :PRINT GR$(J);" ";:
       GR$=GR$(J)
CL 460 IF VAL(GR$)=0 THEN GOSUB 490
JD 470 GR=VAL(GR$)+GR :NEXT:PRINT"AVE";GR/Z
NH 480 NEXT T :GOSUB 2820 :GOTO 50
NF 490 REM CHANGE LETTERS INTO NUMBERS
EG 500 IF GR$= "A+"OR GR$= "a+"THEN GR$="97"
JB 510 IF GR$= "A" OR GR$="a"THEN GR$="93
HF 520 IF GR$= "A-"OR GR$="a-" THEN GR$="89"
QM 530 IF GR$= "B+"OR GR$="b+" THEN GR$="87"
MH 540 IF GR$= "B" OR GR$="b"THEN GR$="83"
QO 550 IF GR$= "B-"OR GR$="b-"THEN GR$="79"
NH 560 IF GR$= "C+" OR GR$= "c+" THEN GR$="77"
HN 570 IF GR$= "C" OR GR$= "c"THEN GR$="73"
AF 580 IF GR$= "C-"OR GR$="c-"THEN GR$="69"
GL 590 IF GR$= "D+"OR GR$= "d+"THEN GR$="67"
EN 600 IF GR$= "D" OR GR$ = "d"THEN GR$="63"
OD 610 IF GR$= "D-"THEN OR GR$ = "d-"GR$="59"
JE 620 IF GR$= "E+"OR GR$= "e+" OR GR$="F+" OR GR
       $= "f+" THEN GR$="54"
IA 630 IF GR$= "E"OR GR$= "e"OR GR$="F"OR GR$= "f
       " THEN GR$= "50"
CH 640 IF GR$= "E-"OR GR$= "e-" OR GR$="F-"OR GR$
        = "f-" THEN GR$="46"
NK 650 RETURN
IF 660 REM read assignments
OM 670 CLS :LOCATE 10,16:PRINT"WORKING":LOCATE 12
       ,11:PRINT"Please be patient"
PP 680  AS =1 :FL =1
PE 690  FOLD$ = "assign"
FD 700  ON ERROR GOTO 740
ED 710  OPEN FOLD$ FOR INPUT AS 1
PN 720  INPUT #1,AS$(AS):AS=AS +1:IF EOF(1)THEN 7
       40
PM 730  GOTO 720
```

```
IL 74Ø  CLOSE 1 :IF ERR = 53 THEN CLS :LOCATE 2,1
        2: PRINT"No previous entries are in this f
        ile.":FOR T=1 TO 3ØØØ:NEXT :RESUME 5Ø  ELS
        E 75Ø
FM 75Ø CLS:X=2:Y=7:LOCATE 3,14: PRINT"Assignments
        ":FOR N = 1 TO AS-1 :LOCATE Y,X:PRINT N;CH
        R$(29);". ";AS$(N):Y=Y+1
JC 76Ø  IF Y=22 THEN Y=7 :GOSUB 282Ø :CLS
GK 77Ø  NEXT N
BO 78Ø GOSUB 282Ø :GOTO 5Ø
HF 79Ø IF R=Ø THEN GOSUB 268Ø:GOSUB 282Ø :ELSE GO
        SUB 277Ø:GOSUB 282Ø
AE 8ØØ CLS
IP 81Ø LOCATE 2,9:PRINT "Adding names to class":L
        OCATE 4,4:PRINT "There are";NS-1;"students
         in this class."
OB 82Ø IF NS=46 THEN LOCATE 12,4:COLOR 23,Ø:PRINT
         "CLASS IS FULL.  NO MORE STUDENTS":COLOR
        7,Ø:GOSUB 282Ø:GOTO 5Ø
EP 83Ø LOCATE 23,4:PRINT "Type XXX for first name
         for MENU"
CH 84Ø LOCATE 6,11:PRINT "Student number"NS":":PR
        INT:INPUT "First name please:";T$
KO 85Ø IF T$="" THEN 79Ø
AB 86Ø IF T$="XXX" OR T$="xxx" THEN 5Ø
MB 87Ø NA$(NS,1)=T$
DO 88Ø PRINT:INPUT "Last name please:";T$
KG 89Ø IF T$="" THEN 79Ø
FG 9ØØ IF T$="XXX" OR T$= "xxx" THEN 5Ø
CP 91Ø NA$(NS,2)=T$:FILE$="stu"+RIGHT$(STR$(NS),L
        EN(STR$(NS))-1)
FN 92Ø OPEN FILE$ FOR OUTPUT AS  1
IC 93Ø WRITE #1,NA$(NS,1):WRITE #1,NA$(NS,2)
FD 94Ø CLOSE #1:NS=NS+1:GOTO 8ØØ
BP 95Ø CLS
QL 96Ø LOCATE 1,9:PRINT "Enter students' grades":
        LOCATE 3,14:PRINT "INSTRUCTIONS"
KF 97Ø PRINT:PRINT "Grades may be any number from
         Ø to 1ØØ  or any letter from A to F.":PRI
        NT
HL 98Ø PRINT "Letter grades may include a plus or
         a   minus.  Letter grades are averaged as
           follows:":PRINT:PRINT
OF 99Ø PRINT "A+=97   A=93   A-=89":PRINT "B+=87
         B=83   B-=79":PRINT "C+=77   C=73   C-=6
        9"
HG 1ØØØ PRINT "D+=67   D=63   D-=59":PRINT "E+=54
          E=5Ø   E-=46":PRINT "F+=54   F=5Ø   F-
         =46"
```

```
DJ 1010 LOCATE 23,9:PRINT "Press any key to begin
        "
HA 1020 IF INKEY$="" THEN 1020
IC 1030 CLS:IF R=0 THEN GOSUB 2680:ON ERROR GOTO
        0 ELSE GOSUB 2770
AB 1040 IF NS = 1 THEN FOR ED = 1 TO 1000:NEXT :G
        OTO 50 ELSE LOCATE 22,7:INPUT"Enter stude
        nt number: ";Y$
HG 1050 IF Y$="XXX" OR Y$ = "xxx" THEN 50
JB 1060 NU=VAL(Y$):IF NU<1 OR NU >M-1 THEN CLS:GO
        SUB 2840 :GOTO 1030
LK 1070 CLOSE 1:FILE$="stu"+RIGHT$(STR$(NU),LEN(S
        TR$(NU))-1):OPEN FILE$ FOR INPUT AS 1
KE 1080 INPUT #1,NA$(NU,1):INPUT #1,NA$(NU,2)
OL 1110 CLOSE 1:OPEN FILE$ FOR APPEND AS  1
KF 1120 CLS:LOCATE 24,9:PRINT"Type XXX when finis
        hed";
MN 1130 LOCATE 2,11:PRINT "ENTER STUDENTS' GRADES
        ":LOCATE 4,11:PRINT STRING$(29,32):LOCATE
         4,7:PRINT "Grade for ";NA$(NU,1);" ";NA$
        (NU,2);
CF 1140 INPUT GA$
HJ 1150 IF GA$="XXX" OR GA$ = "xxx" THEN CLOSE 1:
         GOTO 1030
MJ 1160 IF GA$="" OR GA$= " " THEN 1130
BA 1170 GT = VAL(GA$):IF GT<>0 THEN 1210
MN 1175 IF GA$ ="0"THEN 1210
ON 1180 IF LEN(GA$)>1 AND GA$ <> "0" AND RIGHT$(G
        A$,1)<>"-"AND RIGHT$(GA$,1)<>"+" THEN GOS
        UB 2840 :GOTO 1130
AH 1190 IF ASC(LEFT$(GA$,1))<65 OR ASC(LEFT$(GA$,
        1))> 70 THEN 1200 ELSE 1220
CD 1200 IF ASC(LEFT$(GA$,1))<97 OR ASC(LEFT$(GA$,
        1))>102  THEN GOSUB 2840:GOTO 1130
JE 1210 IF VAL  (GA$)<0 OR VAL(GA$)>100 THEN GOSUB
         2840 :GOTO 1130
MH 1220 LOCATE 18,9:INPUT"Is this grade correct";
        A$
AD 1230 IF A$= ""  THEN 1220
IJ 1240 IF LEFT$(A$,1) = "Y"OR LEFT$(A$,1)= "y" T
        HEN 1270
JM 1250 IF LEFT$(A$,1)="N" OR LEFT$(A$,1)= "n" TH
        EN 1130
LP 1260 GOTO 1130
PO 1270 WRITE #1,GA$:GOTO 1120
GN 1280 CLS :LOCATE 10,16:PRINT"WORKING":LOCATE 1
        2,11:PRINT"Please be patient"
EJ 1290 AS =1 :FL =1
CD 1300 FOLD$ = "assign"
MH 1310 ON ERROR GOTO 1350
```

```
IH 132Ø OPEN FOLD$ FOR INPUT AS 1
BP 133Ø INPUT #1,AS$(AS):AS=AS +1:IF EOF(1)THEN 1
        35Ø
MD 134Ø GOTO 133Ø
FF 135Ø CLOSE 1 :IF ERR = 53 THEN CLS :LOCATE 2,1
        2: PRINT"No previous entries are in this
        file.":FOR T=1 TO 1ØØØ:NEXT :RESUME 14ØØ
        ELSE 136Ø
JO 136Ø CLS:X=2:Y=7:LOCATE 3,14:PRINT"Assignments
        ":FOR N = 1 TO AS-1 :LOCATE Y,X:PRINT N;C
        HR$(29);". ";AS$(N):Y=Y+1
OL 137Ø IF Y=22 THEN Y=7 :GOSUB 282Ø :CLS
KE 138Ø NEXT N
MK 139Ø GOSUB 282Ø
GM 14ØØ ON ERROR GOTO Ø:CLS: GOSUB 286Ø:LOCATE 1,
        1:PRINT "Enter assignment #"AS;:INPUT SA$
PA 141Ø IF SA$= "" OR LEN(SA$)> 28 THEN GOSUB 284
        Ø :GOTO 14ØØ ELSE AS$(AS)= SA$
QL 142Ø IF AS$(AS)<>"XXX" AND AS$(AS)<>"xxx"THEN
        AS = AS+1 :GOTO 14ØØ ELSE OPEN FOLD$ FOR
        OUTPUT AS 1
OA 143Ø FOR T = 1 TO AS-1 :WRITE #1,AS$(T)
DA 144Ø NEXT :CLOSE 1
KJ 145Ø GOTO 5Ø
DF 146Ø  GOSUB 287Ø:GOSUB 282Ø:CLS :LOCATE 7,1Ø:P
        RINT"Do you want to print:"
NP 147Ø LOCATE 1Ø,1Ø:PRINT"1. Grades "
ML 148Ø LOCATE 13,1Ø:PRINT"2. Assignments"
HN 149Ø A$ = INKEY$ :IF A$="" THEN 149Ø
NN 15ØØ IF LEN(A$) = 2 THEN LOCATE 16,1 :PRINT"Us
        e the number keys above the keyboard":FOR
         T= 1 TO 1ØØØ:NEXT:LOCATE 16,1:PRINT STRI
        NG$(4Ø,32):GOTO 149Ø
QL 151Ø IF VAL(A$)=1 THEN 153Ø
PP 152Ø IF VAL (A$)=2 THEN 173Ø ELSE 149Ø
IO 153Ø REM print grades
LD 154Ø ON ERROR GOTO Ø
FO 155Ø IF R=Ø THEN GOSUB 268Ø:ELSE GOSUB 277Ø
CJ 156Ø IF  NS = 1 THEN  LPRINT "No students have
         been entered":FOR T = 1 TO 1ØØØ:NEXT :GO
        TO 5Ø
EH 157Ø  LOCATE 22,1:INPUT "First student number"
        ;FI$:FI= VAL(FI$):IF FI$= "XXX" OR FI$= "
        xxx"THEN 5Ø
MJ 158Ø LOCATE 22,1 :PRINT STRING$(25,32):LOCATE
        22,1 :INPUT"Last student number ";LA$:LA=
        VAL(LA$):IF LA$="XXX"OR LA$= "xxx" THEN 5
        Ø
EA 159Ø IF LA<FI THEN CLS:GOSUB 284Ø:GOTO 153Ø
QK 16ØØ IF LA>NS-1 THEN CLS:GOSUB 284Ø :GOTO 153Ø
```

```
HN 161Ø IF FI<>Ø AND LA <>Ø THEN 162Ø ELSE CLS:GO
        SUB 284Ø:GOTO 153Ø
DH 162Ø CLS:FOR T= FI TO LA :CLOSE 1: FILE$ = "st
        u" +RIGHT$(STR$(T),LEN(STR$(T))-1):OPEN F
        ILE$ FOR INPUT AS  1
BC 163Ø IF T= 1Ø OR T= 2Ø OR T =3Ø OR T=4Ø THEN G
        OSUB 282Ø :CLS
DA 164Ø INPUT #1,NA$(T,1): INPUT #1,NA$(T,2):Z=1:
         REM get both names
LL 165Ø IF EOF(1) THEN GR$(Z)= "No grades" :GOTO
        168Ø
ME 166Ø INPUT #1,GR$(Z):IF EOF(1) THEN 168Ø ELSE
        Z=Z+1:GOTO 166Ø
EI 167Ø RESUME 168Ø
AE 168Ø GR= Ø:CLOSE 1
GH 169Ø LPRINT FILE$;"  ";NA$(T,2);"  ";:FOR J= 1
         TO Z :LPRINT GR$(J);"  ";:GR$=GR$(J)
LD 17ØØ IF VAL(GR$)=Ø THEN GOSUB 49Ø
PE 171Ø GR=VAL(GR$)+GR :NEXT J:LPRINT "AVE";GR/Z
DF 172Ø NEXT T :LPRINT CHR$(12):GOSUB 282Ø :GOTO
        5Ø
GI 173Ø REM print assignments
LI 174Ø CLS:LOCATE 1Ø,16:PRINT "WORKING":LOCATE 1
        2,11:PRINT "Please be patient"
KA 175Ø AS=1:FL=1:ON ERROR GOTO 179Ø
NF 176Ø OPEN "assign" FOR INPUT AS 1
GO 177Ø INPUT #1,AS$(AS):AS=AS+1:IF EOF(1)THEN 17
        9Ø
EL 178Ø GOTO 177Ø
AA 179Ø CLOSE 1 :IF ERR = 53 THEN CLS :LOCATE 2,1
        2: PRINT"No previous entries are in this
        file.":FOR T=1 TO 1ØØØ:NEXT :RESUME 5Ø
PC 18ØØ CLS:LPRINT"Assignments":FOR N = 1 TO AS-1
         :LPRINT N;CHR$(29);". ";AS$(N)
MN 181Ø NEXT N:LPRINT CHR$(12)
IK 182Ø GOSUB 282Ø:GOTO 5Ø
LA 183Ø CLS :LOCATE 7,1Ø: PRINT "Do you wish to c
        hange:"
NO 184Ø LOCATE 1Ø,1Ø:PRINT"1. Grades "
MK 185Ø LOCATE 13,1Ø:PRINT"2. Assignments"
CB 186Ø LOCATE 16,1Ø:PRINT"3. Names"
LB 187Ø A$ = INKEY$ :IF A$="" THEN 187Ø
BN 188Ø IF LEN(A$) = 2 THEN LOCATE 16,1 :PRINT"Us
        e the number keys above the keyboard":FOR
         T= 1 TO 1ØØØ:NEXT:LOCATE 16,1:PRINT STRI
        NG$(4Ø,32):GOTO 187Ø
GD 189Ø IF VAL(A$)=1 THEN 192Ø
EB 19ØØ IF VAL (A$)=2 THEN 219Ø
DE 191Ø IF VAL (A$)= 3 THEN 236Ø ELSE 187Ø
NK 192Ø IF R = Ø THEN GOSUB 268Ø ON ERROR GOTO Ø:
        ELSE GOSUB 277Ø
```

```
MA 193Ø IF NS = 1 THEN FOR T= 1 TO 1ØØØ:NEXT :GOT
        O 5Ø
PC 194Ø LOCATE 23,3 :INPUT"Which student do you w
        ant ";NU$
OP 195Ø NU=VAL(NU$):IF (NU <1 OR NU> NS-1) AND NU
        $<>"XXX" AND NU$<> "xxx" THEN 194Ø
GC 196Ø IF NU$="XXX" OR NU$ = "xxx" THEN CLOSE 1
        :GOTO 5Ø
QP 197Ø CLOSE  1:FILE$="stu"+RIGHT$(STR$(NU),LEN(
        STR$(NU))-1):OPEN FILE$ FOR INPUT AS  1
KE 198Ø I=1:INPUT #1,NA$(NU,1):INPUT #1,NA$(NU,2)
        :IF EOF(1) THEN CLOSE 1:GOTO 2Ø2Ø
KK 199Ø INPUT #1,GR$(I): I=I+1:ON ERROR GOTO 2Ø1Ø
FI 2ØØØ GOTO 199Ø
QK 2Ø1Ø RESUME 2Ø2Ø :ON ERROR GOTO Ø
FF 2Ø2Ø CLS:LOCATE 4,2:FOR T=1 TO 2 :PRINT NA$(NU
        ,T);" ";:NEXT
FO 2Ø3Ø Y=6:X=2:FOR T= 1 TO I-1:LOCATE Y,X:Y=Y+1:
        IF Y = 16 THEN Y=6 :X=X+15
OJ 2Ø4Ø PRINT T;CHR$(29)". "GR$(T):NEXT
CI 2Ø5Ø IF I= 1 THEN LOCATE 12,12:PRINT"No grades
         to edit":FOR T= 1 TO 1ØØØØ:NEXT:GOTO 5Ø
FN 2Ø6Ø GOSUB 286Ø:LOCATE 1,1:PRINT STRING$(36,32
        ):LOCATE 1,1:INPUT"Which grade do you wan
        t to change ";GD$
JP 2Ø7Ø IF GD$="XXX"OR GD$ = "xxx" THEN 5Ø
NF 2Ø8Ø GD =VAL(GD$):IF GD>I-1 OR GD <1 THEN CLS:
        GOSUB 284Ø:GOTO 2Ø2Ø
QP 2Ø9Ø LOCATE 2,1:PRINT "New grade";:INPUT NG$
QL 21ØØ NT = VAL(NG$):IF NT <>Ø THEN 214Ø
KD 211Ø IF LEN(NG$)>1 AND NG$ <> "Ø" AND RIGHT$(N
        G$,1)<>"-"AND RIGHT$(NG$,1)<>"+" THEN CLS
        :GOSUB 284Ø :GOTO 2Ø2Ø
LF 212Ø IF ASC(LEFT$(NG$,1))<65 OR ASC(LEFT$(NG$,
        1))> 7Ø THEN 213Ø ELSE 215Ø
HN 213Ø IF ASC(LEFT$(NG$,1))<97 OR ASC(LEFT$(NG$,
        1))>1Ø2  THEN CLS:GOSUB 284Ø:GOTO 2Ø2Ø
GI 214Ø IF VAL (NG$)<Ø OR VAL(NG$)>1ØØ THEN CLS:G
        OSUB 284Ø :GOTO 2Ø2Ø
MN 215Ø IF NG$="XXX"OR NG$ = "xxx" THEN 5Ø
HK 216Ø GR$(GD)=NG$
PM 217Ø CLOSE 1:OPEN FILE$ FOR OUTPUT AS 1
KG 218Ø FOR T= 1 TO 2 :WRITE #1,NA$(NU,T):NEXT:FO
        R T= 1 TO I-1 :WRITE #1,GR$(T):NEXT:CLOSE
         1 :GOTO 2Ø2Ø
GP 219Ø CLS :LOCATE 1Ø,16:PRINT"WORKING":LOCATE 1
        2,11:PRINT"Please be patient"
EC 22ØØ  AS =1
HE 221Ø  FOLD$ = "assign"
DI 222Ø  ON ERROR GOTO 226Ø
```

```
KA 2230  OPEN FOLD$ FOR INPUT AS 1
EG 2240  INPUT #1,AS$(AS):AS=AS +1:IF EOF(1)THEN
        2260
LC 2250  GOTO 2240
HA 2260  CLOSE 1 :IF ERR = 53 THEN CLS :LOCATE 2,
        12: PRINT"No previous entries are in this
         file.":FOR T=1 TO 1000:NEXT :RESUME 50
        ELSE 2270
OP 2270 CLS:X=2:Y=7:LOCATE 3,14: PRINT"Assignment
        s":FOR N = 1 TO AS-1 :LOCATE Y,X:PRINT N;
        CHR$(29);". ";AS$(N):Y=Y+1
GG 2280  IF Y=22 THEN Y=7:GOSUB 2820:CLS
IN 2290  NEXT N:GOSUB 2820
DN 2300  CLS :GOSUB 2860 :LOCATE 3,1:INPUT"Which
        assignment do you want to change";ASN$
JI 2310 IF ASN$ = "XXX" OR ASN$ = "xxx" THEN 50 E
        LSE ASN = VAL(ASN$) : IF ASN<1 OR ASN >AS
        -1 THEN 2300
BP 2320 LOCATE 10,4 :PRINT"Old assignment name is
         ";AS$(ASN)
CE 2330 LOCATE 15,4:PRINT "New assignment name";:
        INPUT AN$:AS$(ASN)=AN$
JN 2340 OPEN "assign" FOR OUTPUT AS 1
PE 2350 FOR N = 1 TO AS-1 :WRITE #1,AS$(N):NEXT :
        CLOSE 1 : GOTO 2270
EA 2360 REM CHANGE NAMES
CB 2370 CLS:IF R= 0 THEN GOSUB 2680 ELSE GOSUB 27
        70
ME 2380 IF NS = 1 THEN FOR T= 1 TO 1000:NEXT :GOT
        O 50
IJ 2390 LOCATE 23,3:INPUT"Which student (#) do yo
        u want ";NU$:NU=VAL(NU$):IF NU <> 0 AND (
        NU<1 OR NU>NS-1) THEN CLS:GOSUB 2840 :GOT
        O 2370
PK 2400 IF NU$ ="XXX" OR NU$="xxx" THEN 50
BD 2410 CLOSE 1 : FILE$ = "stu" + RIGHT$(STR$(NU)
        ,LEN(STR$(NU))-1):OPEN FILE$ FOR INPUT AS
         1
PB 2420 I=1: INPUT #1,NA$(NU,1):INPUT #1,NA$(NU,2
        ):IF EOF(1) THEN CLOSE 1 :GOTO 2460
DD 2430 INPUT #1,GR$(I):I=I+1:ON ERROR GOTO 2450
ON 2440 GOTO 2430
CB 2450 RESUME 2460:ON ERROR  GOTO 0
ID 2460 CLOSE 1:CLS: LOCATE 4,2:FOR T= 1 TO 2:PRI
        NT NA$(NU,T);" ";:NEXT
GB 2470 GOSUB 2860:LOCATE 8,2:INPUT"New first nam
        e: ";NB1$
HH 2480 LOCATE 11,2:INPUT"New last name: ";NB2$:I
        F NB1$= "XXX"OR NB1$="xxx" OR NB2$= "XXX"
        OR NB2$="xxx" THEN 50
```

```
IK 2490 LOCATE 15,2 :INPUT"Is this information co
        rrect ";A$:IF LEFT$(A$,1)= "y"OR LEFT$(A$
        ,1)= "Y" THEN NA$(NU,1)= NB1$:NA$(NU,2)=
        NB2$:GOTO 2500 ELSE GOTO 2460
PO 2500 OPEN FILE$ FOR OUTPUT AS 1
HN 2510 WRITE #1,NA$(NU,1):WRITE #1,NA$(NU,2)
JL 2520 FOR J = 1 TO I-1 :WRITE #1,GR$(J):NEXT:CL
        OSE 1:GOSUB 2770:GOSUB 2820 :GOTO 50
CJ 2530 CLS:IF R= 0 THEN GOSUB 2680 ELSE GOSUB 27
        70
GA 2540 GOSUB 2820 :CLS
KA 2550 LOCATE 4,15:COLOR 23,0:PRINT "WARNING":CO
        LOR 7,0
LM 2560 LOCATE 10,1:PRINT "This section will crea
        te new files.":PRINT "Be sure a new forma
        tted disk is "
AC 2570 PRINT "in drive A before beginning."
FO 2580 LOCATE 15,11:PRINT "Press 'Y' to begin"
LJ 2590 LOCATE 17,19:PRINT "or":PRINT:PRINT "Type
         XXX if you are not ready to start  to cr
        eate new files on a new disk"
BA 2600 LOCATE 23,1:INPUT A$:IF A$<>"XXX" AND A$<
        > "xxx" AND LEFT$(A$,1)<>"y" AND  LEFT$(A
        $,1) <> "Y" THEN 2600
KH 2610 IF A$="XXX" OR A$ = "xxx" THEN 50
DK 2620 CLOSE 1:FOR Q = 1 TO NS-1
BF 2630 FILE$ = "stu"+RIGHT$(STR$(Q),LEN(STR$(Q))
        -1)
QM 2640 OPEN FILE$ FOR OUTPUT AS 1
HK 2650 WRITE #1,NA$(Q,1):WRITE #1,NA$(Q,2)
KP 2660 CLOSE 1:NEXT Q
NG 2670 CLS:GOSUB 2820 :GOTO 50
LD 2680 CLS:LOCATE 10,16:PRINT "WORKING":LOCATE 1
        2,11:PRINT "Please be patient"
IG 2690 NS=1:R=1
MP 2700 FILE$="STU"+RIGHT$(STR$(NS),LEN(STR$(NS))
        -1)
EI 2710 ON ERROR GOTO 2750
IC 2720 OPEN FILE$ FOR INPUT AS 1:IF EOF(1)THEN 2
        760
DO 2730 INPUT #1,NA$(NS,1):INPUT #1,NA$(NS,2)
BF 2740 CLOSE 1:NS=NS+1:GOTO 2700
DC 2750 RESUME 2760
BJ 2760 CLOSE 1
IE 2770 ON ERROR GOTO 0:CLS:LOCATE 3,5:PRINT"Stud
        ents already in your class"
OC 2780 X=2:Y=6:FOR M= 1 TO NS-1:LOCATE Y,X :PRIN
        T M;CHR$(29);". ";LEFT$(NA$(M,2),8):Y=Y+1
JM 2790 IF Y=21 THEN Y=6:X=X+13
JM 2800 NEXT M
```

```
JP 2810 RETURN
ED 2820 ON ERROR GOTO 0:LOCATE 23,7:PRINT"Press a
        ny key to continue."
AG 2830 A$=INKEY$:IF A$=""THEN 2830 ELSE RETURN
FP 2840 REM ERROR IN INPUT
HJ 2850 LOCATE 15,13:PRINT"IMPROPER INPUT":FOR QR
         = 1 TO 1000:NEXT QR :LOCATE 15,10:PRINT
        STRING$(20,32):RETURN
LD 2860 LOCATE 23,7:PRINT"Type XXX when finished.
        " :RETURN
FD 2870 CLS :LOCATE 11,3 : PRINT "Make sure that
        your printer is online"
KB 2880 LOCATE 13,12: PRINT "before continuing!"
KH 2890 RETURN
```

CHAPTER 5

Graphics

The Screen Machine

Charles Brannon

"The Screen Machine" is a drawing editor that lets you easily draw, color, design, or doodle on your PC or PCjr screen. You can save pictures to disk, even use the screens in your own programs. The PC version requires a color/graphics adapter, advanced BASIC (BASICA), and at least 80K of memory. The PCjr version requires an Enhanced Model with Cartridge BASIC. For both, a joystick is highly recommended.

Your PC or PCjr has a lot of graphics potential. SCREEN 1, the medium-resolution mode, has 320 dots horizontally and 200 vertically. Any dot can be one of four colors. This is as much resolution and more colors than most microcomputer high-resolution modes. In addition, the PCjr has a 320 × 200 mode with a full 16 colors.

IBM's Microsoft BASIC is big and powerful. A full set of commands is available for drawing points, lines, boxes, circles, arcs, and shapes. A powerful PAINT command lets you fill in any shape with any color. PUT and GET let you grab objects from the screen and move them around, even animate them. But you had to be a programmer to make the most of these capabilities—that is, before "The Screen Machine."

Program 1 is the 4-color version of the Screen Machine. It is intended for the IBM PC, but will also work on the PCjr. Program 2 takes advantage of one of the PCjr's special graphics modes to allow 16 colors. Although you can make limited use of the program without a joystick, one is highly recommended.

When you first run the program, you see a brief flurry of screen activity, then you are presented with your drawing screen. At the bottom of the screen is your palette of colors. On the PC, you'll see red, green, and yellow bars. On the PCjr, you'll have 16 colored bars. Above the bars is your crosshair, controlled by the joystick. When you press the red button (button A1 on the top, not the rear of the joystick), a point is drawn in the current color at the center of the crosshair.

At the bottom of the screen, a box encloses the color with which you are drawing. Your crosshair is like a paintbrush, and the colored boxes are your palette. To select a different drawing color, move your crosshair to the color of your choice, then press the joystick button. You have two palette choices on the PC: red, green, and yellow (the default palette); and cyan, purple, and white. You can toggle (switch) between the two palettes by pressing the Enter key. Since the PCjr has all 16 colors available, this command is not in the PCjr version.

Using the Joystick

The IBM joystick lets you instantly move from one area of the screen to another. You can operate it with autocentering, so the stick will spring back to the center position when you let go. This makes it hard to use for drawing, though. If you turn the joystick upside down and hold the stick in the lower-right corner, you can flip two switches to release the centering. The stick can now float and stay wherever you leave it. Try using the joystick in both the free and autocentering modes to see which you prefer.

The response of the joystick is so fast that it can be difficult to finely control the crosshair for delicate shapes. That's why the Screen Machine has another mode of joystick operation, intended to be used with the joystick autocentered. Press the minus key to shift the crosshair into the slow mode. Instead of the stick determining the absolute position of the crosshair, it is now a relative joystick. Push the stick up, and the crosshair will begin to move up. Push it down, and the crosshair moves down. Imagine that the screen is divided into four quadrants: north, south, east, and west. You can also move diagonally.

If you prefer even finer control, you can use the cursor keys to move the crosshair in the slow mode. These cursor controls will work only if you have a joystick plugged in.

To set a point in slow mode, press the period key. To draw a fine line, repeatedly press the period and cursor keys.

If you don't own a joystick, or if you prefer never to use a joystick, see the modifications at the end of this article.

Joystick Calibration

All joysticks are not created equal. The range from east to west and north to south can vary, even with the same joystick. You can adjust a dial or lever to change the horizontal and vertical range, but you still might not be able to reach all areas of the screen. Even worse than not being able to reach all parts of the screen, though, is moving your cursor off the screen—this can cause an Illegal Function Call Error and you may lose your picture. It is therefore important that you calibrate the joystick each time you run the Screen Machine before you begin drawing.

To calibrate your joystick with the Screen Machine, press the J key. A screen message will tell you to move the stick to the lower-right corner of the joystick, then press the joystick button. This sets the upper range. Next, move the stick to the upper-left corner and press the button again to set the lower limit. Now you can control the crosshair over the full range of the screen. Incidentally, some controllers such as a mouse or graphics tablet simulate a joystick, so they may work with this program.

Now you can draw freehand. You may want to play around and get some experience with the joystick. The more you work with the stick, the more control you'll gain.

But there's a lot more to the Screen Machine: It can connect two points to draw a line automatically, draw a rectangle given two opposite corners, or scribe a circle given the center point and a radius. It can fill shapes. It can even let you pick up pieces of the screen, move them around, put them down again, copy them as many times as you want, or store the images on an alternate screen. All these require two points, though, and your crosshair is just one.

Marking the Second Spot

The space bar sets the second mark. The mark is a stationary crosshair. To reset the mark, move the joystick and press the space bar again.

The rectangle, circle, and draw commands require that you set a mark to establish a point of origin. For example, to draw a circle, position the crosshair at the spot you intend to be the center. Press the space bar to mark it. Now move the

crosshair to where the edge of the circle should be (the distance between the two crosshairs will be the radius). Press C, and you have your circle.

Drawing a line is a similar process. Position the crosshair at the point of origin and mark it by pressing the space bar. Then move to the desired endpoint and press D to connect the two crosshairs. The stationary crosshair will remain so that you can press D again with the joystick crosshair in a different place. You can very easily create a drawing with several lines emanating from a single point (the marked spot). You can also draw concentric circles this way.

To draw a box or rectangle, set the mark to represent one corner of the rectangle. Move the crosshair to where the diagonally opposite corner should be, then press R (for Rectangle). Visualization is important with these commands. You should try to imagine the box, line, or circle before you actually create it.

When you begin to use the draw command, you may find yourself resetting the mark position to draw a series of connected lines. Instead, you may prefer a variation of the draw command. Press Ctrl-D (hold down the Control key while pressing D). This will set a mark, as the space bar does. Every time you press the joystick button, the mark is connected to the crosshair with a line, then the mark is reset to the crosshair position. You can then move the crosshair to draw another connected line. Many figures can be made from a series of lines this way. You can press either the space bar or Ctrl-D again to reset the mark position. You know you are in draw mode if there is a capital D in the lower-left corner of the screen.

When you are finished with the mark, you'll probably want to get it off the screen. To do this, press Esc (Escape). This also cancels draw mode.

Fill 'Er Up!

Fill is perhaps the most fun of all the commands. You can easily create abstract art by drawing irregular shapes, then filling them in. You can turn a circle into a solid circle, or an empty box into a solid square. Fill can turn your PC or PCjr screen into a coloring book.

There are just a few things to watch out for, though. First, make sure your figure is entirely enclosed by a line in one color. If there is even a tiny gap in the outline or boundary,

the fill will "escape" from the outline, spilling out to fill the entire screen. This also happens if the boundary line is not all one color. Either way, your picture will be wiped out.

The Fill command involves two colors: the fill color (which is the same as your drawing color) and the boundary color (which may or may not be the same).

Fill works differently on the PC and PCjr versions of the Screen Machine. On the PC, place your crosshair inside the figure you want to fill, then press F. You'll be asked which color is the boundary. Press one letter: (R)ed, (G)reen, (Y)ellow, or (B)ackground. If you are using the alternate color set, you'll press either (C)yan, (P)urple, (W)hite, or (B)ackground. This tells the Fill command where to stop. If you don't tell it the right boundary color, it will keep going until it finds the color you've actually pressed.

Since the PCjr version has too many colors to spell out in a prompt, you select the boundary color in a different way. Remember that in both versions, a box surrounds the color bar of the current drawing color. In the PCjr version, the boundary color is indicated by a horizontal line bisecting the corresponding color bar (by default, black).

There are two ways to set the boundary color. If the crosshair is somewhere on the drawing screen, press F to set the boundary color to your drawing color. This is convenient if you are filling a figure with the same color as the outlines of the figure.

Another way to set the boundary color is to move the crosshair to the color bars, just as if you were selecting another drawing color. But instead of pressing the joystick button, press F to select that color as the boundary color. Now you can move back up to somewhere inside the boundary of the figure and press Ctrl-F to do the fill.

Alternate Screens

But wait! You have not just one screen to draw on, but two. Press the Tab key (the one with two arrows on it), and the current screen will be swapped with the alternate one. Of course, the current screen is now the alternate screen, and vice versa.

On the PCjr version of the Screen Machine, you can also copy the current screen to the alternate screen by pressing Shift-Tab. That way, you can periodically save the screen

you're working on to the alternate page, so in case you accidentally mess up your picture, you can bring back the earlier version. There's another powerful use of this screen, which we'll cover in a moment.

Grab a Grape

Sometimes, drawing can be tediously repetitive. For example, when drawing a bunch of grapes, instead of drawing each grape, wouldn't it be easier to draw just one "perfect" grape, then copy it over and over again to build up the whole bunch? The most flexible and useful commands in the Screen Machine are GET and PUT. They let you pick up (GET) any rectangular area of the screen and then deposit it (PUT) anywhere else. It's like a rubber stamp.

To GET a shape, you have to define a rectangular area. This is similar to drawing a rectangle. Use the space bar to mark one corner of the area, then move the crosshair to the opposite corner. Press G, and suddenly your crosshair becomes the area you've defined. You can move the shape anywhere, just like your crosshair. In fact, the shape you've picked up *is* your crosshair. You can still draw with it. The bigger the shape, though, the slower the movement. You cannot pick up a shape bigger than an area equivalent to one-fourth the screen. You could, though, have a shape that is one-eighth the screen width, and a whole screen high (since this equals one-fourth the screen). Bigger shapes will have to be moved a piece at a time (like those ancient castles which are relocated in sections, then reassembled).

Once you've picked up a shape, you can lay it down with PUT. First, press P. You'll see this prompt:

PUT: (P)SET,P(R)ESET,(A)ND,(O)R,(X)OR

These options specify exactly how the image will be rubber-stamped on the screen. If you press P for (P)SET, the shape is simply placed on the screen. Any blank areas in the shape will erase whatever they are laid upon.

R for P(R)ESET will lay down an inverted image of the shape—all the colors are reversed.

A for (A)nd is tricky. It will PUT down only those parts of the shape that coincide with whatever's underneath the shape.

O for (O)r merges the shape with the background. Black areas are ignored, but some colors may be changed, since PUT works on the binary level.

X for (X)OR creates a composite image of the shape and the background. Parts of the image will be reversed according to the color of the background. The unique feature of XOR is that if you do it twice, the background is restored.

After you've picked up a shape, it still remains on the screen. GET copies the rectangular area, but does not remove it. To remove a shape, GET the shape and PUT it back in exactly the same spot with XOR. It will be neatly excised. You'll have to do a lot of experimenting to fully appreciate all the variations of the PUT command.

Computer-Aided Design?

With the alternate screen, a new application opens up. On one screen you can store a variety of templates, such as logic or electronics symbols, drafting diagrams, chemistry apparatus, and so on. You can then assemble circuits or lab equipment by picking up shapes from one page, then laying them down on the other page. You're going from the general to the specific. There are many practical applications of this technique, which is a simple form of Computer-Aided Design (CAD).

When you are finished with your shape, press Backspace to discard it. The crosshair is back, and you can pick up another shape or start drawing again.

There are so many ways you can take advantage of the various commands. It's like working with the English language, or BASIC—using a fundamental vocabulary to create sonnets, or drawing programs. The same concept applies here: With a few basic commands, you can build intricate drawings.

For example, how would you draw a half-circle? There are several ways. Draw a circle, then use GET to remove the top half. Or draw a circle so that half of it is drawn off the screen, then GET it and PUT it where you want. How many other ways can you think of? Which is easiest for you?

You can draw a solid box in either yellow (PC) or bright white (PCjr), then GET it. Now you have a box that when you PUT it down with PRESET will erase whatever's under it. This is an easy way to erase parts of the screen. Sometimes when I'm doing detailed work, I'll pick up a single dot and use that as my crosshair.

Preserving for Posterity

If you put in as much time as I have working on your pictures, you'll want a permanent record. One way is to dump it to the printer. Before you load BASIC from DOS, type GRAPHICS with the DOS system master disk in the drive. This will change the way the PrtSc (Print Screen) key works. It can now copy a graphics screen to the printer. Just draw your masterpiece, and press Shift-PrtSc.

Of course, you can also save your picture to disk. Press S, and give the filename. If you don't specify an extender, .PIC will be added. If you get an error, try again with a different disk (the one you used may be full or write-protected). Recalling a picture is just as simple. Press L, give the filename, and watch the screen come in. It's an interesting effect, like venetian blinds: First one half of the screen loads, then the other half is interleaved with it. This is due to the organization of screen memory. On the PCjr, this happens in four stages. Picture files made with the PC and PCjr versions are not compatible. The 4-color pictures are 16K long; the 16-color ones require 32K.

You can easily load the screens from your own BASIC program. You could design an intricate background for a game with the Screen Machine, then load it in from your program with BLOAD. Use this line:

```
DEF SEG=&HB800:BLOAD "name",0
```

Of course, you need to first set up the graphics screen with SCREEN 1 on the PC. On the PCjr (the 16-color mode) SCREEN 5 requires that you put a CLEAR ,,,32768 at the top of your program, before any variables are defined. This allocates the extra memory required by SCREEN 5.

Miscellaneous Commands

Ctrl-Home clears the screen (Ctrl-Fn-Home on the PCjr). Press B to change the background color. Press X to toggle the color bars on/off. To remove the crosshair, just GET an empty piece of background.

You can easily add text with the T command. Position the crosshair where you want the text to be drawn, then press T. Enter the text, then press Enter. The text will be copied to the screen in the current drawing color. Be careful, though, because the input prompt at the bottom of the screen cannot prevent you from cursoring around or clearing the screen.

Finally, when you are through with the Screen Machine, press Q to quit the program and return to BASIC.

If you're going crazy trying to memorize all these commands, press H (or any noncommand key), and you can browse through the help screen, one command at a time (it's really a help *line*). To make things really easy, you can refer to the handy quick-reference chart below. Photocopy it, glue it to an index card, and place it near your work area as you're learning to use the program.

Quick-Reference Chart

B	Change Background color
C	Circle from mark (center) to radius (crosshair)
D	Draw line from mark to crosshair
F	Fill (PC version)
F	Select Fill boundary color (PCjr version)
Ctrl-F	Fill in area (PCjr version)
G	Get shape from mark to crosshair (opposite corner)
H	View Help screen
J	Joystick calibration
L	Load picture
P	PUT shape
Q	Quit program
R	Rectangle from mark to opposite corner (crosshair)
S	Save picture
T	Enter Text at crosshair
X	Toggle color bars on/off
Space bar	Set start position (mark)
Esc	Escape from draw mode/cancel mark
+ key	Fast joystick mode
− key	Slow joystick mode
Enter	Switch palettes (PC version only)
Ctrl-Home	Clear screen (erase picture)
Tab	Switch screens
Shift-Tab	Copy from current to alternate page (PCjr version only)
Backspace	Discard shape
Cursor keys	Move crosshair in slow mode
Period (.)	Plot a single point

Without a Joystick

If you want to use the Screen Machine without a joystick, make the following modifications to either version of the program:

```
240 GOTO 310
320 IF DRAWMODE THEN IF TR THEN
    IF Y<185 AND OY<185 THEN GOSUB 1010:
    LINE(SAVX,SAVY)-(X+3,Y+3),C:
    SAVX=X+3:SAVY=Y+3:GOSUB 1010:TR=0:
    GOTO 340
330 IF TR THEN IF Y<185 AND OY<185 THEN
    TR=0:LINE (OX+3,OY+3)-(X+3,Y+3),C
    ELSE C=INT ((X+3)/80):GOSUB 980
355 IF C$="," THEN TR=1:GOTO 310
```

These changes will cause the program to ignore the joystick. Use the cursor keys to move the crosshair. Where the program previously requires you to press the fire button (such as in Draw mode), use the comma (,) key instead.

Since it takes so long to move the crosshair across the screen with the cursor keys, add this line to the PC version:

```
357 IF C$>="0" AND C$<="3"
    THEN C=VAL (C$):GOSUB 980:
    GOTO 310
```

or this line to the PCjr version:

```
357 IF C$=">" THEN C=C=(C+1) and 15:
    GOSUB 980:GOTO 310
```

Now on the PC you can directly access any color by pressing a number key from 0 to 3. And on the PCjr you can advance the drawing color by pressing the greater than (>) key. You may want to use this modification even if you do use a joystick.

Program 1. Four-Color Screen Machine (PC Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
CD 1ØØ 'The Screen Machine PC Version
OO 11Ø 'requires color adaptor
EN 12Ø 'and advanced BASIC (BASICA)
EI 13Ø DEFINT A-Y:SCREEN 1:CLS:KEY OFF:STRIG ON:D
       IM SHAPE(2ØØ4)
DD 14Ø GOSUB 128Ø:CLS:DEF SEG:CALL Z
GO 15Ø COLOUR$="Green Red Yellow Cyan Purple Whit
       e"
GM 16Ø XMIN=3:YMIN=3:XMAX=138:YMAX=12Ø
AA 17Ø CLS:CB=1:C=2:BC=-1:COLOR Ø,Ø,Ø
QI 18Ø FX!=312/(XMAX-XMIN):FY!=192/(YMAX-YMIN)
KL 19Ø XWIDTH=8:YLIMIT=8:XLIMIT=32Ø-8:YLIMIT=2ØØ-
       8
```

```
ON 2ØØ LINE (3,Ø)-(3,6),3:LINE (Ø,3)-(6,3),3:PRES
       ET(3,3)
OG 21Ø DIM CROSS(1Ø):GET (Ø,Ø)-(7,7),CROSS
OF 22Ø GOSUB 98Ø
NE 23Ø ON ERROR GOTO Ø
JP 24Ø SØ=STICK(Ø):S1=STICK(1):TR=STRIG(1):IF JOY
       MODE=Ø THEN 29Ø
OL 25Ø DX=4Ø-SØ:DY=4Ø-S1
HI 26Ø X=X+(SØ<3Ø AND X>Ø)-(SØ>5Ø AND X<XLIMIT)
OK 27Ø Y=Y+(S1<3Ø AND Y>Ø)-(S1>5Ø AND Y<YLIMIT)
NB 28Ø OSØ=SØ:OS1=S1:GOTO 31Ø
NE 29Ø X=FX!*SØ-XMIN:Y=FY!*S1-YMIN:IF X>XLIMIT TH
       EN X=XLIMIT
PA 3ØØ IF Y>YLIMIT THEN Y=YLIMIT
JB 31Ø IF OTHER THEN PUT(OX,OY),SHAPE ELSE PUT (O
       X,OY),CROSS
MK 32Ø IF DRAWMODE THEN IF TR THEN IF Y<185 AND O
       Y<185 THEN GOSUB 1Ø1Ø:LINE (SAVX,SAVY)-(X+
       3,Y+3),C:SAVX=X+3:SAVY=Y+3:GOSUB 1Ø1Ø:GOTO
        34Ø
AD 33Ø IF TR THEN IF Y<185 AND OY<185 THEN LINE (
       OX+3,OY+3)-(X+3,Y+3),C ELSE C=INT((X+3)/8Ø
       ):GOSUB 98Ø
KF 34Ø IF OTHER THEN PUT(X,Y),SHAPE ELSE PUT (X,Y
       ),CROSS
AB 35Ø OX=X:OY=Y:C$=INKEY$:IF C$="" THEN 24Ø
ON 36Ø IF C$>="a" AND C$<="z" THEN C$=CHR$(ASC(C$
       )-32)
CP 37Ø IF C$<>CHR$(Ø)+CHR$(119) THEN 4ØØ ELSE MSG
       $="Erase picture -- Are you sure (Y/N):":G
       OSUB 1Ø3Ø
CJ 38Ø IF RTN$="y" OR RTN$="Y" THEN CLS:GOSUB 98Ø
       :GOSUB 1ØØØ:GOTO 24Ø
EG 39Ø GOTO 24Ø
GD 4ØØ IF C$="-" THEN JOYMODE=1:GOTO 24Ø
DB 41Ø IF C$="+" THEN JOYMODE=Ø:GOTO 24Ø
IA 42Ø IF C$=CHR$(9) THEN GOSUB 1ØØØ:DEF SEG:V=VA
       RPTR(ML$):Z=FNML!(Ø):CALL Z:GOSUB 98Ø:GOSU
       B 1ØØØ:GOTO 24Ø
MB 43Ø IF C$=CHR$(Ø)+CHR$(72) THEN Y=Y+(Y>Ø):GOTO
        31Ø
AA 44Ø IF C$=CHR$(Ø)+CHR$(8Ø) THEN Y=Y-(Y<YLIMIT)
       :GOTO 31Ø
KG 45Ø IF C$=CHR$(Ø)+CHR$(75) THEN X=X+(X>Ø):GOTO
        31Ø
BJ 46Ø IF C$=CHR$(Ø)+CHR$(77) THEN X=X-(X<XLIMIT)
       :GOTO 31Ø
DI 47Ø IF C$="." THEN IF Y<185 THEN PSET(X+3,Y+3)
       ,C:GOTO 24Ø
```

```
KB 480 IF C$=CHR$(13) THEN CMODE=1-CMODE:COLOR,CM
       ODE:GOTO 240
GD 490 IF C$="B" THEN BK=(BK+1)AND 15:COLOR BK:GO
       TO 240
LA 500 IF C$=" " OR C$=CHR$(4) THEN IF SAVX THEN
       GOSUB 1010:SAVX=0
OD 510 IF C$=" " OR C$=CHR$(4) THEN IF SAVX=0 AND
        Y<180 THEN SOUND 100,3:SAVX=X+3:SAVY=Y+3:
       GOSUB 1010 ELSE BEEP:GOTO 240
QO 515 IF C$=" " THEN 240
CH 520 IF C$=CHR$(27) THEN IF SAVX THEN GOSUB 101
       0:SAVX=0:DRAWMODE=0:GOSUB 980:GOTO 240 ELS
       E 240
OH 530 IF C$=CHR$(8) THEN GOSUB 1000:XLIMIT=312:Y
       LIMIT=192:OTHER=0:GOSUB 1000:GOTO 240
NM 540 IF C$<>"J" THEN 680
FJ 550 GOSUB 1000:T=CB:CB=0:GOSUB 980:LOCATE 25,1
       :PRINT"Hold stick to lower right, press FI
       RE";:XMAX=0:YMAX=0
IN 560 IF STICK(0)>XMAX THEN XMAX=STICK(0)
OP 570 IF STICK(1)>YMAX THEN YMAX=STICK(1)
HB 580 IF STRIG(1)=0 THEN 560
KE 590 IF STRIG(1) THEN 590
MI 600 GOSUB 980:LOCATE 25,1:PRINT"Hold stick to
       upper left, press FIRE";
LP 610 XMIN=1000:YMIN=1000
OC 620 IF STICK(0)<XMIN THEN XMIN=STICK(0)
EE 630 IF STICK(1)<YMIN THEN YMIN=STICK(1)
CO 640 IF STRIG(1)=0 THEN 620
FH 650 IF STRIG(1) THEN 650
QJ 660 FX!=312/(XMAX-XMIN):FY!=192/(YMAX-YMIN)
DP 670 CB=T:GOSUB 980:GOSUB 1000:GOTO 240
JE 680 IF C$<>"C" THEN 710 ELSE IF SAVX=0 THEN BE
       EP:GOTO 240
OM 690 DX=SAVX-X-3:DY=SAVY-Y-3:R=SQR(DX*DX+DY*DY)
CN 700 GOSUB 1000:CIRCLE (SAVX,SAVY),R,C:GOSUB 98
       0:GOSUB 1000:GOTO 240
IM 710 IF C$=CHR$(4) THEN DRAWMODE=1:SOUND 500,2:
       GOSUB 980:GOTO 240
MB 715 IF C$="D" THEN IF Y<188 AND SAVX THEN GOSU
       B 1000:LINE (SAVX,SAVY)-(X+3,Y+3),C:GOSUB
       1000:GOTO 240 ELSE BEEP:GOTO 240
HN 720 IF C$="R" THEN IF Y<185 AND SAVX THEN GOSU
       B 1000:LINE (SAVX,SAVY)-(X+3,Y+3),C,B:GOSU
       B 1000:GOTO 240 ELSE BEEP:GOTO 240
BH 730 IF C$<>"G" THEN 750 ELSE IF SAVX=0 THEN BE
       EP:GOTO 240
PF 732 XWIDTH=ABS(SAVX-X-3):YWIDTH=ABS(SAVY-Y-3):
       IF XWIDTH=0 OR YWIDTH=0 THEN BEEP:GOTO 240
```

```
DK 735 IF 4+INT((XWIDTH*2+9)/8)*(YWIDTH+1)>4004 T
       HEN BEEP:GOTO 240
BO 740 GOSUB 1000:GET(X+3,Y+3)-(SAVX,SAVY),SHAPE
IB 745 OTHER=1:XLIMIT=319-XWIDTH:YLIMIT=199-YWIDT
       H
MI 746 IF X>XLIMIT THEN X=XLIMIT:OX=X
EN 747 IF Y>YLIMIT THEN Y=YLIMIT:OY=Y
DN 749 GOSUB 1000:GOTO 240
OA 750 IF C$<>"P" THEN 830 ELSE IF OTHER=0 THEN B
       EEP:GOTO 240
DH 760 MSG$="PUT:(P)SET,P(R)ESET,(A)ND,(O)R,(X)OR
       ":GOSUB 1030
JG 770 IF RTN$<>"" THEN V=INSTR("PRAXO",RTN$):IF
       V THEN ON V GOTO 780,790,800,810,820
FF 775 GOTO 240
DE 780 GOSUB 1000:PUT (X,Y),SHAPE,PSET:GOSUB 1000
       :GOTO 240
FC 790 GOSUB 1000:PUT (X,Y),SHAPE,PRESET:GOSUB 10
       00:GOTO 240
EO 800 GOSUB 1000:PUT (X,Y),SHAPE,AND:GOSUB 1000:
       GOTO 240
CN 810 GOSUB 1000:PUT (X,Y),SHAPE,XOR:GOSUB 1000:
       GOTO 240
MK 820 GOSUB 1000:PUT (X,Y),SHAPE,OR:GOSUB 1000:G
       OTO 240
AC 830 IF C$="X" THEN CB=1-CB:GOSUB 980:GOTO 240
MF 840 IF C$<>"F" THEN 880
PK 850 THIS$=MID$(COLOUR$,17*CMODE+1,17):MSG$="Bo
       undary color? Bkgrd "+THIS$:GOSUB 1030:IF
       RTN$="" OR RTN$=" " THEN 870
OC 860 BC=INT((INSTR(THIS$,RTN$)+4)/5):GOSUB 1000
       :PAINT (X+3,Y+3),C,BC:BC=-1:GOSUB 1000
BI 870 GOSUB 980:GOTO 240
EE 880 IF C$="T" THEN IF Y<185 THEN MSG$="Text:":
       GOSUB 1340:GOSUB 1000:LOCATE Y/8+1,X/8+1:D
       EF SEG:POKE &H4E,C:PRINT RTN$;:GOSUB 1000:
       POKE &H4E,3:GOTO 240 ELSE BEEP:GOTO 240
LN 890 IF C$="Q" THEN MSG$="Quit: Are you sure? (
       Y/N):":GOSUB 1030:IF RTN$="Y" THEN SCREEN
       0,0,0:END ELSE 240
CE 900 IF C$<>"S" THEN 940
NA 910 MSG$="Save picture: Filename?":GOSUB 1340:
       IF RTN$="" THEN 240
GE 920 GOSUB 1000:ON ERROR GOTO 925:DEF SEG=&HB80
       0:BSAVE RTN$+".PIC",0,&H4000:ON ERROR GOTO
        0:GOSUB 1000:GOTO 240
OD 925 GOSUB 1000
HN 930 MSG$="Error...Press ENTER:":GOSUB 1030:RES
       UME 230
BD 940 IF C$<>"L" THEN 970
```

```
BE 950 MSG$="Load picture: Filename?":GOSUB 134Ø:
       IF RTN$="" THEN 24Ø
CA 96Ø ON ERROR GOTO 93Ø:DEF SEG=&HB8ØØ:BLOAD RTN
       $+".PIC",Ø:GOSUB 98Ø:GOSUB 1ØØØ:GOTO 23Ø
PO 97Ø MSG$="Need help? (Y/N):":GOSUB 1Ø3Ø:IF RTN
       $="y" OR RTN$="Y" THEN 1Ø7Ø ELSE 24Ø
OL 98Ø LINE (Ø,188)-(319,199),Ø,BF:IF CB THEN FOR
        I=Ø TO 3:LINE (I*8Ø+2,19Ø)-(I*8Ø+77,197),
       I,BF:NEXT:LOCATE 25,2:PRINT CHR$(32+36*DRA
       WMODE);:LINE (C*8Ø,188)-(C*8Ø+79,199),3,B
NF 99Ø RETURN
FG 1ØØØ IF OTHER THEN PUT (X,Y),SHAPE ELSE PUT (X
        ,Y),CROSS
HM 1Ø1Ø IF SAVX THEN PUT(SAVX-3,SAVY-3),CROSS,XOR
IB 1Ø2Ø RETURN
BC 1Ø3Ø GOSUB 1ØØØ:T=CB:CB=Ø:GOSUB 98Ø:LOCATE 25,
        1:PRINT MSG$;:RTN$=INPUT$(1):CB=T:GOSUB 9
        8Ø:GOSUB 1ØØØ
LC 1Ø4Ø IF RTN$>="a" AND RTN$<="z" THEN RTN$=CHR$
        (ASC(RTN$)-32)
EG 1Ø5Ø IF RTN$=CHR$(13) THEN RTN$=""
JN 1Ø6Ø RETURN
EB 1Ø7Ø MSG$="HELP: Press RETURN to continue...":
        TT=CB:CB=Ø:GOSUB 1Ø3Ø:RESTORE 1Ø7Ø:GOSUB
        1ØØØ
KG 1Ø8Ø READ MSG$:IF MSG$="END" THEN CB=TT:GOSUB
        98Ø:GOSUB 1ØØØ:GOTO 24Ø
NM 1Ø9Ø GOSUB 98Ø:LOCATE 25,2Ø-LEN(MSG$)/2:PRINT
        MSG$;
FB 11ØØ IF INKEY$="" THEN 11ØØ
ND 111Ø GOTO 1Ø8Ø
PM 112Ø DATA {-} Slow cursor,{+} Fast Cursor
HJ 113Ø DATA {Ctrl-Home} Clear screen
HO 114Ø DATA {B} Change background color
MC 115Ø DATA {Enter} Toggle color palette
KG 116Ø DATA SPACE BAR  Mark start position
EG 117Ø DATA {Esc} Cancel mark/draw mode
AA 1175 DATA {D} Draw line from mark
DN 118Ø DATA {Ctrl-D} Enter draw mode
GL 119Ø DATA {C} Draw circle Mark=Center
KI 12ØØ DATA {R} Draw rectangle Mark=Opposite cor
        ner
EF 121Ø DATA {X} Toggle color bars on/off
KD 122Ø DATA {F} Fill in area
DP 123Ø DATA {T} Enter text at crosshair
PH 124Ø DATA {Q} Quit program
EG 125Ø DATA {S} Save picture
PF 126Ø DATA {L} Load picture
HD 127Ø DATA {Tab} Switch to alternate page
KK 1271 DATA {G} Get shape
```

```
JB 1272 DATA {P} Put shape
QM 1273 DATA {Backspace} Discard shape
FF 1274 DATA {J} Joystick calibration
GI 1275 DATA Cursor keys move cursor in slow mode
MO 1276 DATA {.} Plot point
FB 1277 DATA END
PG 128Ø DEF SEG:ML$=SPACE$(39):V=VARPTR(ML$):DEF
        FNML!(DUMMY)=PEEK(V+1)+256*PEEK(V+2)
EC 129Ø RESTORE 132Ø:Z=FNML!(Ø):FOR I=Ø TO 38:REA
        D A:CKSUM=CKSUM+A:POKE Z+I,A:NEXT
BD 13ØØ IF CKSUM=3842 THEN RETURN
BG 131Ø SCREEN Ø,Ø,Ø:COLOR 31:PRINT"Error";:COLOR
         7:PRINT" in DATA statements.":END
BP 132Ø DATA 85,3Ø,19Ø,Ø,Ø,187,Ø,16,142,219,139,4
        ,187,Ø,184,142,219,135,4,187,Ø,16,142,219
GO 133Ø DATA 137,4,7Ø,7Ø,129,254,Ø,64,114,227,31,
        93,2Ø2,Ø,Ø
GP 134Ø GOSUB 1ØØØ:T=CB:CB=Ø:GOSUB 98Ø:LOCATE 25,
        1:PRINT MSG$;:LINE INPUT ;RTN$:CB=T:GOSUB
         98Ø:GOSUB 1ØØØ:RETURN
```

Program 2. The Screen Machine with 16 Colors (PCjr Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
GB 1ØØ 'The Screen Machine (PCjr)
CL 125 CLEAR,,,65536!
MA 13Ø DEFINT A-Y:SCREEN 5:COLOR 7,Ø:CLS:KEY OFF:
       STRIG ON:DIM SHAPE(4ØØ4)
JP 14Ø C=1:CB=1:SCREEN ,,1:CLS:GOSUB 98Ø:SCREEN ,
       ,Ø:CLS
HD 16Ø XMIN=3:YMIN=3:XMAX=11Ø:YMAX=119
EF 18Ø FX!=319/(XMAX-XMIN):FY!=192/(YMAX-YMIN)
HN 19Ø XWIDTH=7:YWIDTH=7:YLIMIT=8:XLIMIT=32Ø-7:YL
       IMIT=2ØØ-7
ML 2ØØ LINE (3,Ø)-(3,6),15:LINE (Ø,3)-(6,3),15:PR
       ESET(3,3)
DI 21Ø DIM CROSS(18):GET (Ø,Ø)-(6,6),CROSS
OF 22Ø GOSUB 98Ø
NE 23Ø ON ERROR GOTO Ø
JP 24Ø SØ=STICK(Ø):S1=STICK(1):TR=STRIG(1):IF JOY
       MODE=Ø THEN 29Ø
II 26Ø X=X+(SØ<2Ø AND X>Ø)-(SØ>6Ø AND X<XLIMIT)
BK 27Ø Y=Y+(S1<2Ø AND Y>Ø)-(S1>6Ø AND Y<YLIMIT)
NB 28Ø OSØ=SØ:OS1=S1:GOTO 31Ø
GI 29Ø X=FX!*(SØ-XMIN):Y=FY!*(S1-YMIN):IF X>XLIMI
       T THEN X=XLIMIT
PA 3ØØ IF Y>YLIMIT THEN Y=YLIMIT
```

```
JB 310 IF OTHER THEN PUT(OX,OY),SHAPE ELSE PUT (O
       X,OY),CROSS
MK 320 IF DRAWMODE THEN IF TR THEN IF Y<185 AND O
       Y<185 THEN GOSUB 1010:LINE (SAVX,SAVY)-(X+
       3,Y+3),C:SAVX=X+3:SAVY=Y+3:GOSUB 1010:GOTO
        340
QJ 330 IF TR THEN IF Y<185 AND OY<185 THEN LINE (
       OX+3,OY+3)-(X+3,Y+3),C ELSE C=INT((X+3)/20
       ):GOSUB 980
KF 340 IF OTHER THEN PUT(X,Y),SHAPE ELSE PUT (X,Y
       ),CROSS
AB 350 OX=X:OY=Y:C$=INKEY$:IF C$="" THEN 240
ON 360 IF C$>="a" AND C$<="z" THEN C$=CHR$(ASC(C$
       )-32)
CP 370 IF C$<>CHR$(0)+CHR$(119) THEN 400 ELSE MSG
       $="Erase picture -- Are you sure (Y/N):":G
       OSUB 1030
CJ 380 IF RTN$="y" OR RTN$="Y" THEN CLS:GOSUB 980
       :GOSUB 1000:GOTO 240
EG 390 GOTO 240
GD 400 IF C$="-" THEN JOYMODE=1:GOTO 240
DB 410 IF C$="+" THEN JOYMODE=0:GOTO 240
PP 415 IF C$="F" THEN IF Y<188 THEN FC=C ELSE FC=
       INT((X+3)/20)
LJ 417 IF C$="F" THEN GOSUB 1000:GOSUB 980::GOSUB
        1000:GOTO 240
EG 420 IF C$=CHR$(9) THEN GOSUB 1000:SCR=1-SCR:SC
       REEN ,,SCR,SCR,0:GOSUB 980:GOSUB 1000:GOTO
        240
NO 425 IF C$=CHR$(0)+CHR$(15) THEN GOSUB 1000:ss
       i SCR,1-SCR:GOSUB 1000:GOTO 240
MB 430 IF C$=CHR$(0)+CHR$(72) THEN Y=Y+(Y>0):GOTO
        310
AA 440 IF C$=CHR$(0)+CHR$(80) THEN Y=Y-(Y<YLIMIT)
       :GOTO 310
KG 450 IF C$=CHR$(0)+CHR$(75) THEN X=X+(X>0):GOTO
        310
BJ 460 IF C$=CHR$(0)+CHR$(77) THEN X=X-(X<XLIMIT)
       :GOTO 310
DI 470 IF C$="." THEN IF Y<185 THEN PSET(X+3,Y+3)
       ,C:GOTO 240
FP 490 IF C$="B" THEN BK=(BK+1)AND 15:COLOR,BK:GO
       TO 240
LA 500 IF C$=" " OR C$=CHR$(4) THEN IF SAVX THEN
       GOSUB 1010:SAVX=0
OD 510 IF C$=" " OR C$=CHR$(4) THEN IF SAVX=0 AND
        Y<180 THEN SOUND 100,3:SAVX=X+3:SAVY=Y+3:
       GOSUB 1010 ELSE BEEP:GOTO 240
QO 515 IF C$=" " THEN 240
```

```
CH 520 IF C$=CHR$(27) THEN IF SAVX THEN GOSUB 101
       0:SAVX=0:DRAWMODE=0:GOSUB 980:GOTO 240 ELS
       E 240
BD 530 IF C$=CHR$(8) THEN GOSUB 1000:XLIMIT=313:Y
       LIMIT=192:OTHER=0:GOSUB 1000:GOTO 240
NM 540 IF C$<>"J" THEN 680
AD 550 T=CB:CB=0:GOSUB 1000:GOSUB 980:LOCATE 25,1
       :PRINT"Hold stick to lower right, press FI
       RE";:XMAX=0:YMAX=0
IN 560 IF STICK(0)>XMAX THEN XMAX=STICK(0)
OP 570 IF STICK(1)>YMAX THEN YMAX=STICK(1)
HB 580 IF STRIG(1)=0 THEN 560
KE 590 IF STRIG(1) THEN 590
MI 600 GOSUB 980:LOCATE 25,1:PRINT"Hold stick to
       upper left, press FIRE";
LP 610 XMIN=1000:YMIN=1000
OC 620 IF STICK(0)<XMIN THEN XMIN=STICK(0)
EE 630 IF STICK(1)<YMIN THEN YMIN=STICK(1)
CO 640 IF STRIG(1)=0 THEN 620
FH 650 IF STRIG(1) THEN 650
EG 660 FX!=319/(XMAX-XMIN):FY!=192/(YMAX-YMIN)
DP 670 CB=T:GOSUB 980:GOSUB 1000:GOTO 240
JE 680 IF C$<>"C" THEN 710 ELSE IF SAVX=0 THEN BE
       EP:GOTO 240
OM 690 DX=SAVX-X-3:DY=SAVY-Y-3:R=SQR(DX*DX+DY*DY)
CN 700 GOSUB 1000:CIRCLE (SAVX,SAVY),R,C:GOSUB 98
       0:GOSUB 1000:GOTO 240
IM 710 IF C$=CHR$(4) THEN DRAWMODE=1:SOUND 500,2:
       GOSUB 980:GOTO 240
MB 715 IF C$="D" THEN IF Y<188 AND SAVX THEN GOSU
       B 1000:LINE (SAVX,SAVY)-(X+3,Y+3),C:GOSUB
       1000:GOTO 240 ELSE BEEP:GOTO 240
HN 720 IF C$="R" THEN IF Y<185 AND SAVX THEN GOSU
       B 1000:LINE (SAVX,SAVY)-(X+3,Y+3),C,B:GOSU
       B 1000:GOTO 240 ELSE BEEP:GOTO 240
BH 730 IF C$<>"G" THEN 750 ELSE IF SAVX=0 THEN BE
       EP:GOTO 240
PF 732 XWIDTH=ABS(SAVX-X-3):YWIDTH=ABS(SAVY-Y-3):
       IF XWIDTH=0 OR YWIDTH=0 THEN BEEP:GOTO 240
DK 735 IF 4+INT((XWIDTH*4+11)/8)*(YWIDTH+1)>8004
       THEN BEEP:GOTO 240
BO 740 GOSUB 1000:GET(X+3,Y+3)-(SAVX,SAVY),SHAPE
IB 745 OTHER=1:XLIMIT=319-XWIDTH:YLIMIT=199-YWIDT
       H
MI 746 IF X>XLIMIT THEN X=XLIMIT:OX=X
EN 747 IF Y>YLIMIT THEN Y=YLIMIT:OY=Y
DN 749 GOSUB 1000:GOTO 240
OA 750 IF C$<>"P" THEN 830 ELSE IF OTHER=0 THEN B
       EEP:GOTO 240
```

```
DH 76Ø MSG$="PUT:(P)SET,P(R)ESET,(A)ND,(O)R,(X)OR
       ":GOSUB 1Ø3Ø
JG 77Ø IF RTN$<>"" THEN V=INSTR("PRAXO",RTN$):IF
       V THEN ON V GOTO 78Ø,79Ø,8ØØ,81Ø,82Ø
FF 775 GOTO 24Ø
DE 78Ø GOSUB 1ØØØ:PUT (X,Y),SHAPE,PSET:GOSUB 1ØØØ
       :GOTO 24Ø
FC 79Ø GOSUB 1ØØØ:PUT (X,Y),SHAPE,PRESET:GOSUB 1Ø
       ØØ:GOTO 24Ø
EO 8ØØ GOSUB 1ØØØ:PUT (X,Y),SHAPE,AND:GOSUB 1ØØØ:
       GOTO 24Ø
CN 81Ø GOSUB 1ØØØ:PUT (X,Y),SHAPE,XOR:GOSUB 1ØØØ:
       GOTO 24Ø
MK 82Ø GOSUB 1ØØØ:PUT (X,Y),SHAPE,OR:GOSUB 1ØØØ:G
       OTO 24Ø
AC 83Ø IF C$="X" THEN CB=1-CB:GOSUB 98Ø:GOTO 24Ø
GL 84Ø IF C$<>CHR$(6) THEN 88Ø
HB 86Ø GOSUB 1ØØØ:PAINT (X+3,Y+3),C,FC:GOSUB 1ØØØ
BI 87Ø GOSUB 98Ø:GOTO 24Ø
PA 88Ø IF C$="T" THEN IF Y<185 THEN MSG$="Text:":
       GOSUB 134Ø:GOSUB 1ØØØ:LOCATE Y/8+1,X/8+1:C
       OLOR C-(C=Ø),BK:PRINT RTN$;:COLOR 7,BK:GOS
       UB 1ØØØ:GOTO 24Ø ELSE BEEP:GOTO 24Ø
LN 89Ø IF C$="Q" THEN MSG$="Quit: Are you sure? (
       Y/N):":GOSUB 1Ø3Ø:IF RTN$="Y" THEN SCREEN
       Ø,Ø,Ø:END ELSE 24Ø
CE 9ØØ IF C$<>"S" THEN 94Ø
NA 91Ø MSG$="Save picture: Filename?":GOSUB 134Ø:
       IF RTN$="" THEN 24Ø
FF 92Ø GOSUB 1ØØØ:ON ERROR GOTO 925:DEF SEG=&H1ØØ
       Ø+&H8ØØ*SCR:BSAVE RTN$+".PIC",Ø,&H8ØØØ:ON
       ERROR GOTO Ø:GOSUB 1ØØØ:GOTO 24Ø
OD 925 GOSUB 1ØØØ
HN 93Ø MSG$="Error...Press ENTER:":GOSUB 1Ø3Ø:RES
       UME 23Ø
BD 94Ø IF C$<>"L" THEN 97Ø
BE 95Ø MSG$="Load picture: Filename?":GOSUB 134Ø:
       IF RTN$="" THEN 24Ø
ON 96Ø ON ERROR GOTO 93Ø:DEF SEG=&H1ØØØ+&H8ØØ*SCR
       :BLOAD RTN$+".PIC",Ø:GOSUB 98Ø:GOSUB 1ØØØ:
       GOTO 23Ø
PO 97Ø MSG$="Need help? (Y/N):":GOSUB 1Ø3Ø:IF RTN
       $="y" OR RTN$="Y" THEN 1Ø7Ø ELSE 24Ø
OE 98Ø LINE (Ø,188)-(319,199),Ø,BF:IF CB THEN FOR
        I=Ø TO 15:LINE (I*2Ø+2,19Ø)-(I*2Ø+17,197)
       ,I,BF:NEXT:LOCATE 25,1:PRINT CHR$(32+36*DR
       AWMODE);:LINE (C*2Ø,188)-(C*2Ø+19,199),15,
       B:LINE (FC*2Ø,194)-(FC*2Ø+19,194),15
NF 99Ø RETURN
```

```
FG 1000 IF OTHER THEN PUT (X,Y),SHAPE ELSE PUT (X
        ,Y),CROSS
HM 1010 IF SAVX THEN PUT(SAVX-3,SAVY-3),CROSS,XOR
IB 1020 RETURN
BC 1030 GOSUB 1000:T=CB:CB=0:GOSUB 980:LOCATE 25,
        1:PRINT MSG$;:RTN$=INPUT$(1):CB=T:GOSUB 9
        80:GOSUB 1000
LC 1040 IF RTN$>="a" AND RTN$<="z" THEN RTN$=CHR$
        (ASC(RTN$)-32)
EG 1050 IF RTN$=CHR$(13) THEN RTN$=""
JN 1060 RETURN
PD 1070 RESTORE 1115:GOSUB 1000:TT=CB:CB=0
KG 1080 READ MSG$:IF MSG$="END" THEN CB=TT:GOSUB
        980:GOSUB 1000:GOTO 240
NM 1090 GOSUB 980:LOCATE 25,20-LEN(MSG$)/2:PRINT
        MSG$;
FB 1100 IF INKEY$="" THEN 1100
ND 1110 GOTO 1080
MP 1115 DATA "HELP: Press any key to continue
PM 1120 DATA {-} Slow cursor,{+} Fast Cursor
HJ 1130 DATA {Ctrl-Home} Clear screen
HO 1140 DATA {B} Change background color
KG 1160 DATA SPACE BAR  Mark start position
EG 1170 DATA {Esc} Cancel mark/draw mode
AA 1175 DATA {D} Draw line from mark
DN 1180 DATA {Ctrl-D} Enter draw mode
GL 1190 DATA {C} Draw circle Mark=Center
KI 1200 DATA {R} Draw rectangle Mark=Opposite cor
        ner
EF 1210 DATA {X} Toggle color bars on/off
KK 1215 DATA {F} Select fill border color
JN 1220 DATA {Ctrl-F} Fill in area
DP 1230 DATA {T} Enter text at crosshair
PH 1240 DATA {Q} Quit program
EG 1250 DATA {S} Save picture
PF 1260 DATA {L} Load picture
IE 1265 DATA {Tab} Switch to alternate page
DI 1270 DATA {Shift-Tab} Copy to alternate page
KK 1271 DATA {G} Get shape
JB 1272 DATA {P} Put shape
OM 1273 DATA {Backspace} Discard shape
FF 1274 DATA {J} Joystick calibration
GI 1275 DATA Cursor keys move cursor in slow mode
MO 1276 DATA {.} Plot point
FB 1277 DATA END
GP 1340 GOSUB 1000:T=CB:CB=0:GOSUB 980:LOCATE 25,
        1:PRINT MSG$;:LINE INPUT ;RTN$:CB=T:GOSUB
         980:GOSUB 1000:RETURN
```

Custom Characters

Sheldon Leemon

One of the most powerful graphics features on home computers is the ability to design custom character sets. Now you can learn this technique on the IBM PC and PCjr—opening the door to special foreign language characters, mathematical symbols, and easily animated shapes for games. It requires a PC with the color/graphics adapter or a PCjr (128K RAM recommended).

Have you ever seen a text adventure game that displayed its screen messages in Old English lettering? Or a foreign language program that used unusual punctuation symbols and characters which normally aren't available on your IBM keyboard? Or a program which displayed the special symbols of higher mathematics? Or a game that filled the screen with animated aliens and spaceships?

All of these graphics effects, and more, are made possible by one relatively simple programming technique—*custom character sets*. Sometimes the technique is referred to as *programmable characters* or *redefinable characters*. No matter what you call it, the technique is a highly useful one. It can be achieved on almost all home computers, including the Atari, Commodore 64 and VIC-20, Texas Instruments TI-99/4A, and others. Fortunately, the IBM PC and PCjr also have this capability, although not in text mode (the characters must be displayed in a graphics mode—SCREEN 1 or higher).

When you redefine a character, you change its shape into almost anything you want. Then when you print the character on the screen, your custom shape appears instead of the standard shape which was built into the computer. For instance, you could create your own alphabet of Old English lettering—or modern computer-style lettering. You can design special characters and symbols for algebra and trigonometry or foreign languages or programming languages such as APL. You can even design aliens, spaceships, and other shapes for games.

The technique isn't too difficult if you have at least some understanding of BASIC programming. But before we get down to the nuts and bolts, you'll need some background on

how a computer (and specifically, an IBM computer) displays a character on the screen. We'll also cover some fundamentals of the binary number system.

Characters Are Bit Patterns

If you look very closely at the characters displayed on your screen, you'll notice that they are formed of tiny dots. (If your computer is hooked up to an ordinary TV set, which has less resolution than a monitor, the dots may be blurred together and impossible to see.) The arrangement of these dots determines the shape of each character. To redesign a character, you have to rearrange the dots.

The dot patterns for the standard set of characters built into the computer are permanently stored in a ROM (Read Only Memory) chip. As you probably know, ROM is permanent memory which cannot be altered by the user, unlike RAM (Random Access Memory). Forget about rearranging the dot patterns in ROM. The trick to custom characters is to create your own dot patterns in RAM, and then to direct the computer to print those characters on the screen instead of the standard set in ROM.

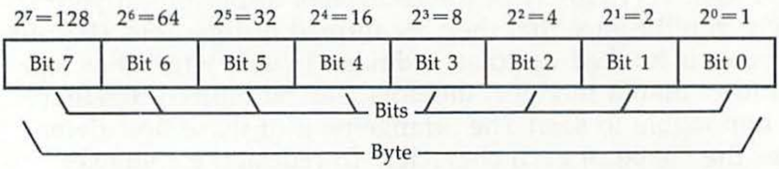
Whether stored in ROM or RAM, the dot patterns are described to the computer by lists of numbers. It takes eight numbers to describe the dot pattern for each character. When the computer is told to display a character (either by someone pressing a key or by a PRINT statement in a BASIC program), the computer's operating system fetches the correct list of eight numbers from memory, translates the numbers into a dot pattern, and displays the dots (and therefore the character) on the screen. The dots are known as a *bit pattern*.

Let's review the binary number system for a moment. A bit is a binary digit, a 1 or a 0. It takes eight bits to make one *byte*. Eight of these bytes, in turn, are the eight numbers which describe the bit pattern for each character.

Each bit in a byte is assigned a value which corresponds to a power of two (see Figure 1). The lowest bit position has a value of 2 to the power of 0 (2^0), which equals 1. The next bit position has a value of 2^1, which equals 2. The third bit is 2^2, which equals 4; the fourth bit is 2^3, which equals 8; and so on, up to 2^7, which equals 128. Each bit in a byte can be either set (turned on), in which case it has the value of its bit position, or reset (turned off), in which case it has a value of 0. If all the

bits in a byte are set, the byte equals 255 (1+2+4+8+16 +32+64+128). By setting and resetting various bits, a byte can equal any number between 0 and 255.

Figure 1. Values of Bit Positions



While all of this might seem confusing if you're unfamiliar with binary arithmetic, it will make more sense when you start designing custom characters. And it's really all the binary you have to know for this technique.

Making Shapes from Bits

Here's why it's important to understand about bits and bytes: Each bit that is set represents one dot in a character on the screen. If a bit is on, a dot appears in that bit position. If a bit is off, no dot appears in that bit position. Since there are eight bits in a byte, and eight bytes per character, a character is formed from a bit pattern of 64 dots arranged in an 8×8 grid (see Figure 2).

Figure 2. The 8×8 Character Grid

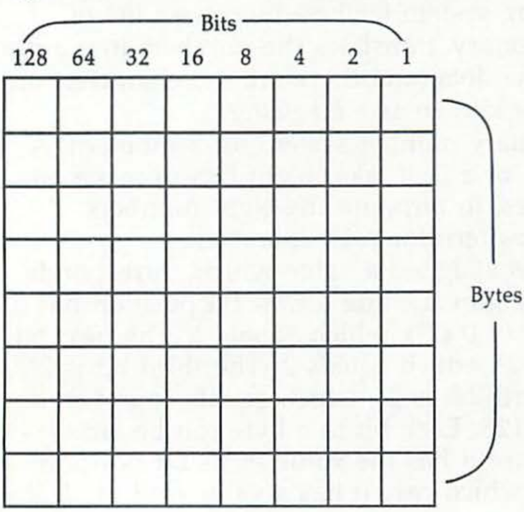
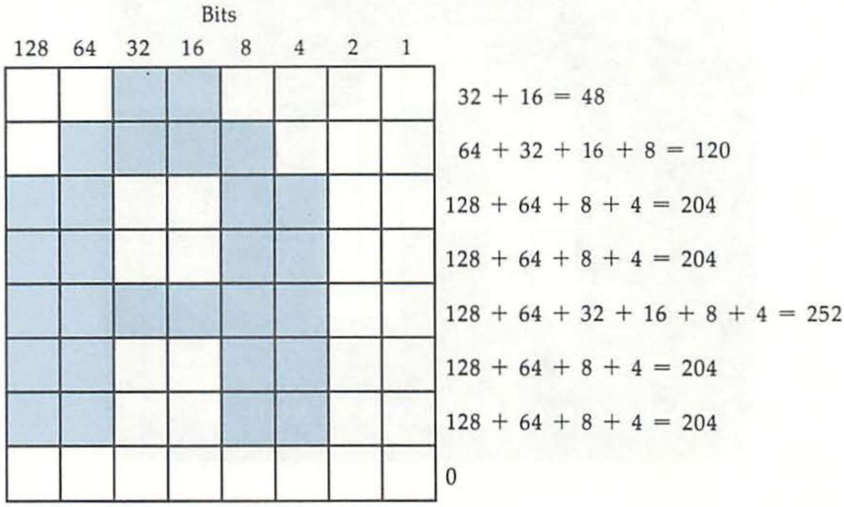


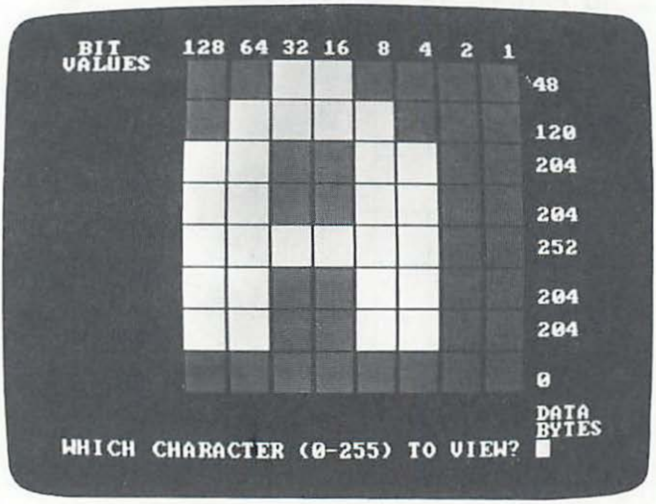
Figure 3. Bit Pattern for the Letter *A*



To derive the list of eight numbers which describe the dot pattern of a character, you must add up all the set bits in each of the eight bytes. It's not as difficult as it sounds. For example, Figure 3 shows how the letter *A* could be designed. Notice how the bit positions of each byte are added up to yield the list of eight numbers.

Program 1, "Viewchar," lets you graphically display the bit patterns of the IBM's standard character set. The character set consists of 256 characters numbered from 0 to 255. When you run Viewchar, it asks you to enter a character number. Then it displays the corresponding bit pattern—"blown up," if you will. Bits that are set (on) appear as white squares, and bits that are reset (off) appear as purple squares (see photo).

To design your own custom characters, you can use graph paper marked off with an 8 × 8 grid. Then color in the spaces to form your character. By adding up the bit position values (refer to Figure 1), you can arrive at the byte values for each row. A much easier way to design custom characters is to use a utility program known as a *character editor*. Character editors let you draw the character on the screen and then calculate the byte values for you.



Here's how the dot pattern for the letter *A* appears when enlarged with the "Viewchar."

Fooling the Computer

After you've designed a set of custom characters, the next step is to store the lists of numbers in RAM and then have the computer look there for its character information instead of in ROM.

If you have at least 128K of RAM in your PC or PCjr, the BASIC reference manual states that memory above the 96K mark is unused by BASIC. That makes it a safe place to store the custom character data. To reach this area with PEEK and POKE statements, the program must define this segment with the statement DEF SEG=&H1700.

To have the computer fetch the character data from that segment, you must change a few memory locations called *pointers*. Pointers let the operating system know where the character data is located. These pointers can be found among the interrupt vectors at the bottom of memory. Each pointer is four bytes long, consisting of a two-byte offset address and a two-byte segment address. The pointer to data for the built-in graphics and foreign language characters (numbered 128–255) in both the PC and PCjr is at interrupt vector 1F (memory locations &H7C, &H7D, &H7E, and &H7F when expressed in

hexadecimal notation). In addition, the PCjr has a pointer to data for characters 0–127, which includes the alphanumeric characters. This pointer is at interrupt vector 44 (memory locations &H110, &H111, &H112, and &H113). Since these pointers are at the bottom of memory, to reach them with PEEK and POKE you must first define the segment with DEF SEG=0.

To get a better idea of what's going on, type in and run Program 2, "Compuset," and Program 3, "Walkman." They show practical examples of custom characters. To maintain compatibility with both the PC and PCjr, these programs redefine only the characters numbered 128–255, but PCjr users should remember that the lower set of characters (0–127) in their computers can also be changed.

The program Compuset shows how to install a custom character font. Interestingly, the bit patterns for this font were transferred to the IBM directly from an Atari computer. Atari, Commodore, and TI computers all use 8 × 8 grids for their characters, just like the IBM. As a matter of fact, you can even use character editor programs on those computers to design custom characters for your PC or PCjr.

Here's how Compuset works. First, the default pointers are saved in the array OLDVEC so the program can switch back to the normal character set when it's done. Next, the segment is set for unused memory above the 96K mark with the statement DEF SEG=&H1700, and the character numbers are read from DATA statements and POKEd into memory at that segment. The pointer to characters 128–255 is set to &H1700 by first changing the segment with DEF SEG=0, and then POKEing three zeros into memory locations &H7C–&H7E, and POKEing &H17 into memory location &H7F.

When the screen is set up for graphics with a SCREEN 1 command, the program can PRINT the new characters. After restoring the pointer by POKEing the contents of the array SAVPNTR back into locations &H110–&H113, the program can print normally once again. It's very important to restore the pointers to their default values after you're finished printing custom characters. If the program ends with the screen set up for graphics, and if your altered character set has replaced the lower set of alphanumeric characters 0–127, the computer will not be able to recognize the custom characters, and it won't be able to respond to any further commands.

Creating Pictures and Animation

If you tried running Compuset, you probably noticed the short pause before the new font appears on the screen. It takes time for the computer to read the character numbers in the DATA statements and POKE them into memory. Luckily, there's a faster way.

After you've run the program once and stored the character numbers in memory, you can save the data to disk with this compound statement:

```
DEF SEG=&H1700:BSAVE "filename",0,1024
```

Then you can replace the READ statement and DATA lines in the program with this statement:

```
DEF SEG=&H1700:BLOAD "filename",0
```

If you're writing a program for publication, of course, you have to incorporate the numbers in DATA statements to make a publishable listing.

Besides creating fancy text fonts, custom characters can also be used to make pictures which are one or more characters in size. You can display these pictures merely by PRINTing the characters on the screen. Pictures made up of several characters can be PRINTed all at once by positioning the cursor with control characters such as CHR$(28), CHR$(29), CHR$(30), and CHR$(31). For example, to print the characters ABC directly over DEF, you could use this statement:

```
PRINT "ABC"+CHR$(29)+CHR$(29)+CHR$(29)+
CHR$(31)+"DEF"
```

Printing groups of characters rapidly in sequence allows you to achieve some degree of animation. Program 3, Walkman, is a good example. It redefines the 26 characters of the alphabet to represent the figure of a man in five different positions. The characters needed to PRINT these pictures are stored in the array MAN$(0)–MAN$(4). By flipping through these pictures, the man appears to walk across the screen.

Overcoming Problems

There are limitations to this kind of animation, however. You'll notice that an elaborate method is used in line 240 to position each picture of the man. This is because when the graphics screen is set up, the LOCATE, TAB, and SPC commands will not allow you to position a character string near

the end of the line if that string will wrap around that line. If it does wrap around, the string is started at the beginning of the next line, despite all efforts to position it near the end of the current line. The only way I've found to print a long string near the end of a line is to pad the string with leading spaces. Unfortunately, because each of these spaces must be printed, the man walks slower as he reaches the edge of the screen.

Another limitation is that a character can be printed in only one color. Normally, text is printed on the graphics screen using the highest color attribute possible. For example, SCREEN 1 uses palette 3, which defaults to white. There is, however, a tricky POKE which can force BASIC to PRINT using one of the other palettes. This allows you to PRINT character graphics in a palette color different from normal text. Use this statement:

DEF SEG:POKE &H43,palette number

This causes the following PRINT statements to display text using the palette you specified. But be careful not to POKE this memory location with a zero. That would PRINT all letters in the background color, rendering them invisible to both you and the computer. Unless such a POKE is changed back by your program, you will have no choice but to switch the computer off and on again to regain control.

Program 1. Viewchar

```
BL 10 DIM MASK(7):FOR BIT=0 TO 7:MASK(BIT)=2^(7-B
      IT):NEXT
MK 20 DEF SEG=&HF000:IF PEEK(&HFFFE)=253 THEN JR=
      -1 ELSE JR=0
DL 30 SCREEN 1: CLS:KEY OFF:DEF SEG = 0
BB 40 OFFSET.HI=PEEK(&H7C)+256*PEEK(&H7D)
DA 50 SEGMENT.HI=PEEK(&H7E)+256*PEEK(&H7F)
AF 60 IF JR THEN OFFSET.LO=PEEK(&H110)+256*PEEK(&
      H111) ELSE OFFSET.LO=&HFA6E
IB 70 IF JR THEN SEGMENT.LO=PEEK(&H112)+256*PEEK(
      &H113) ELSE SEGMENT.LO=&HF000
JL 80 LOCATE 25,1:INPUT"WHICH CHARACTER (0-255) T
      O VIEW";CHOICE
KD 90 OFFSET = OFFSET.LO: DEF SEG = SEGMENT.LO
PF 100 IF CHOICE>127 THEN CHOICE=CHOICE-128:OFFSE
       T=OFFSET.HI:DEF SEG = SEGMENT.HI
OJ 110 CLS:PRINT" BIT      128 64 32 16  8  4  2  1
       ":PRINT"VALUES":ROW=1
DM 120 FOR BYTE=0 TO 7
```

```
DI 130  NUMBER = PEEK(OFFSET+BYTE+8*CHOICE)
JH 140  ROW =ROW+2+(BYTE AND 1): LOCATE ROW,33:PR
       INT NUMBER
DN 150  FOR BIT = 0 TO 7
EH 160    X1=24*BIT+66:Y1=20*BYTE+10
KN 170    X2=X1+21: Y2=Y1+18
GO 180    DOT = (NUMBER AND MASK(BIT)) = 0
KN 190    LINE (X1,Y1)-(X2,Y2),3+DOT,BF
IJ 200 NEXT BIT,BYTE
FO 210 PRINT:PRINT TAB(34)"DATA":PRINT TAB(34)"BY
       TES";:GOTO 80
```

Program 2. Compuset

```
FA 100 DEF SEG = 0
GF 110 FOR VECTOR = 0 TO 3: OLDVEC(VECTOR) = PEEK
       (&H7C+VECTOR):NEXT
LN 120 DEF SEG = &H1700
FD 130 FOR DOTPOS=0 TO 207: READ DOTDATA: POKE DO
       TPOS, DOTDATA:NEXT
GI 140 DEF SEG = 0
IJ 150 MESSAGE$="THIS IS IN COMPUTER SCRIPT"
FK 160 GOSUB 500:SCREEN 1:CLS
IE 170 FOR I=1 TO LEN(MESSAGE$):CHAR= ASC(MID$(ME
       SSAGE$,I,1)) :IF CHAR>63 THEN CHAR=CHAR+63
HE 180 PRINT CHR$(CHAR);:NEXT:PRINT
MD 190 PRINT:FOR I=128 TO 153:PRINT CHR$(I);:NEXT
       :PRINT:PRINT
NB 200 PRINT:PRINT"THESE ARE NORMAL TEXT CHARACTE
       RS"
GC 210 PRINT:FOR I=65 TO 90:PRINT CHR$(I);:NEXT
OP 220 GOSUB 600:END
AH 500 FOR VECTOR = 0 TO 2:POKE (&H7C+VECTOR),0:N
       EXT: POKE &H7F, &H17:RETURN
AD 600 FOR VECTOR = 0 TO 3: POKE (&H7C+VECTOR),OL
       DVEC(VECTOR): NEXT:RETURN
NM 1000 DATA 63,51,51,127,115,115,115,0,126,102,1
        02,127,103,103,127,0,127,103,103,96,99,99
        ,127,0
KO 1010 DATA 126,102,102,119,119,119,127,0,127,96
        ,96,127,112,112,127,0,127,96,96,127,112,1
        12,112,0
HC 1020 DATA 127,99,96,111,103,103,127,0,115,115,
        115,127,115,115,115,0,127,28,28,28,28,28,
        127,0
AA 1030 DATA 12,12,12,14,14,110,126,0,102,102,108
        ,127,103,103,103,0,48,48,48,112,112,112,1
        26,0
```

```
0F 1040 DATA 103,127,127,119,103,103,103,0,103,11
        9,127,111,103,103,103,0,127,99,99,103,103
        ,103,127,0
GF 1050 DATA 127,99,99,127,112,112,112,0,127,99,9
        9,103,103,103,127,7,126,102,102,127,119,1
        19,119,0
HC 1060 DATA 127,96,127,3,115,115,127,0,127,28,28
        ,28,28,28,28,0,103,103,103,103,103,103,12
        7,0
EG 1070 DATA 103,103,103,103,111,62,28,0,103,103,
        103,111,127,127,103,0,115,115,115,62,103,
        103,103,0
HG 1080 DATA 103,103,103,127,28,28,28,0,127,102,1
        08,24,55,103,127,0
```

Program 3. Walkman

```
FA 100 DEF SEG = 0
GF 110 FOR VECTOR = 0 TO 3: OLDVEC(VECTOR) = PEEK
       (&H7C+VECTOR):NEXT
LN 120 DEF SEG = &H1700
FD 130 FOR DOTPOS=0 TO 207: READ DOTDATA: POKE DO
       TPOS, DOTDATA:NEXT
GI 140 DEF SEG = 0
PI 150 BACK$=CHR$(31)+CHR$(29)+CHR$(29)+CHR$(29)
BF 160 BLANK$="
               ":REM This line consists of 41 spac
       es
PA 170 DIM MAN$(4):MAN$(0)="  "+CHR$(128)+BACK$+"
        "+CHR$(129)+CHR$(130)+BACK$+" "+CHR$(131)
       +CHR$(132)
HM 180 MAN$(1)="  "+CHR$(133)+BACK$+" "+CHR$(134)
       +CHR$(135)+BACK$+" "+CHR$(136)+CHR$(137)
HH 190 MAN$(2)="  "+CHR$(138)+BACK$+" "+CHR$(139)
       +CHR$(140)+BACK$+" "+CHR$(141)+CHR$(142)
PI 200 MAN$(3)=" "+CHR$(143)+CHR$(144)+BACK$+" "+
       CHR$(145)+CHR$(146)+BACK$+" "+CHR$(147)+"
       "
EA 210 MAN$(4)=" "+CHR$(148)+CHR$(149)+BACK$+" "+
       CHR$(150)+CHR$(151)+BACK$+" "+CHR$(152)+CH
       R$(153)
OA 220 SCREEN 1:CLS:GOSUB 500
EP 230 FOR REPEAT = 0 TO 6:FOR FRAME = 0 TO 4:LOC
       ATE 10,1
QJ 240 MAN$=LEFT$(BLANK$,FRAME+5*REPEAT)+MAN$(FRA
       ME):PRINT MAN$
CJ 250 GOSUB 700: NEXT FRAME, REPEAT: END: GOSUB
       600:END
```

```
OK 499 END
AH 5ØØ FOR VECTOR = Ø TO 2:POKE (&H7C+VECTOR),Ø:N
       EXT: POKE &H7F, &H17:RETURN
AD 6ØØ FOR VECTOR = Ø TO 3: POKE (&H7C+VECTOR),OL
       DVEC(VECTOR): NEXT:RETURN
EB 7ØØ FOR I= 1 TO 6Ø/(REPEAT+1):NEXT I:RETURN
FG 1ØØØ REM position 1
FB 1Ø1Ø DATA Ø,Ø,Ø,Ø,Ø,56,124,124
ND 1Ø2Ø DATA Ø,Ø,Ø,1,3,15,29,123
QI 1Ø3Ø DATA 124,126,12Ø,224,224,24Ø,24Ø,251
MA 1Ø4Ø DATA 115,7,7,62,252,224,96,48
QK 1Ø5Ø DATA 255,22Ø,192,224,96,118,6Ø,24
NK 1Ø6Ø REM
ID 1Ø7Ø REM position 2
HF 1Ø8Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,28,62
DA 1Ø9Ø DATA Ø,Ø,Ø,Ø,1,7,31,59
FF 2ØØØ DATA 62,62,63,252,224,24Ø,24Ø,24Ø
DD 2Ø1Ø DATA 51,55,7,15,253,24Ø,192,96
BM 2Ø2Ø DATA 254,222,192,128,192,224,96,124
MC 2Ø3Ø REM
JJ 2Ø4Ø REM position 3
DO 2Ø5Ø DATA Ø,Ø,Ø,Ø,Ø,112,248,248
ID 2Ø6Ø DATA Ø,Ø,Ø,1,3,15,31,31
PD 2Ø7Ø DATA 248,252,24Ø,192,192,224,224,224
DA 2Ø8Ø DATA 7,15,111,255,217,195,3,3
MI 2Ø9Ø DATA 24Ø,112,Ø,128,192,128,Ø,192
ML 21ØØ REM
JF 211Ø REM position 4
KM 212Ø DATA Ø,Ø,Ø,Ø,1,3,3,3
GE 213Ø DATA Ø,Ø,Ø,Ø,192,224,224,224
IJ 214Ø DATA 3,3,7,15,31,63,63,31
JJ 215Ø DATA 24Ø,192,Ø,Ø,Ø,128,128,192
BG 216Ø DATA 28,6Ø,6Ø,62,3Ø,3Ø,124,127
NA 217Ø REM
MN 218Ø REM position 5
QP 219Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,1,1
LM 22ØØ DATA Ø,Ø,Ø,Ø,Ø,224,24Ø,24Ø
OC 221Ø DATA 1,1,1,3,7,31,63,63
IH 222Ø DATA 24Ø,248,224,128,128,192,192,192
AE 223Ø DATA 7,15,15,31,61,115,113,6Ø
NG 224Ø DATA 24Ø,112,Ø,128,192,192,24Ø,Ø
```

Creating Animation with PUT and GET

Charles Brannon

GET and PUT are powerful BASIC commands that let you pick up and move pieces of the screen. They are the fundamental tools for animation on the IBM PC and PCjr, and the basis for games in BASIC. If the IBM manual leaves you confused, however, here's all you need to know to GET started.

Your IBM PC or PCjr is a fine graphics system. The PC has a high-resolution, 4-color mode with 320 individual points horizontally and 200 points vertically. The PCjr has a similar mode with 16 colors, as well as a 4-color 640 × 200 mode. IBM BASIC has a very powerful subset of graphics commands that let you draw points, lines, circles, boxes, and figures. There is even a PAINT command that fills in any enclosed area. After working with many BASICs that lack such commands, they are a blessing indeed.

On the other hand, some lower-cost home computers have video features that are a generation beyond the bit-mapped high-resolution displays found on the PC and PCjr. Computers such as the Commodore 64, Atari, Coleco Adam, and TI-99/4A have independently movable display objects called *sprites*. Sprites are like miniature bitmaps that exist on a different plane from the screen text and graphics. They can move in front of or behind other images on the screen without disturbing anything. Since they are displayed by the video hardware instead of by software, they can also move very fast and smoothly. Advanced computer animation depends on such objects, whether they are supported in hardware or simulated by software.

If you've played some of the better videogames on the PC or PCjr, you probably wouldn't notice the lack of sprites. These game programs achieve animation by first drawing the object on the screen, then erasing it (and restoring the background), and finally redrawing the shape at a new location.

This process—draw/erase/redraw (which is supported in hardware on computers with true sprites)—must be done entirely by the game program. Even if the program is written in machine language, which is very fast, there is still a limit to the size and number of shapes you can smoothly animate when using this method.

Simulating Sprites in BASIC

Despite these drawbacks, IBM BASIC does include two commands that can be used to simulate sprite animation: GET and PUT. With careful programming, you can achieve acceptable animation in BASIC with these commands.

Animation is not restricted to games, of course. GET and PUT are generally useful whenever you want to pick up a piece of the high-resolution screen and stamp it elsewhere. It can be very handy for icon-based menus in which users move a crosshair to select symbols representing program options. You can move the crosshair with PUT and even rearrange the symbols. With just a few basic shapes, you can build large pictures, such as customized computer-generated electronic schematics or architectural layouts.

Although the manuals might seem confusing, GET and PUT are really very easy to use and understand. If you've used the Box variation of the LINE command on a Graphics screen, you know that you can draw a box by specifying two diagonally opposite corners:

```
LINE (0,0)-(19,9),1,B
```

This draws a box 20 pixels long and 10 pixels high. That's right—20×10, not 19×9. The point (0,0) has to be counted as well.

To create a movable object, you first have to draw it, then define it. There are two ways to do this. The most commonly used method among IBM programmers is to draw the shape on the screen with the various graphics commands (LINE, DRAW, PAINT, and so forth), and then GET the shape (pick it up) by specifying two diagonally opposite coordinates that would frame it.

For example, the statement GET (0,0),(19,9),D would grab anything within the frame from 0 to 19 across and 0 to 9 down. The picture is automatically converted from pixels to bytes and placed in the array D. You must put the shape into

a numeric array. The array can be single precision, double precision, or integer. I prefer to use an integer array, since you can examine the bytes retrieved by looking at the contents of the array. If you use a floating-point array, the values in the array will appear to be strange, almost random. You can specify an integer array by placing a percent sign (%) after the array name.

Before you can GET a shape, though, you have to DIMension an array large enough to hold the object. IBM gives us a lovely formula, which you'll soon commit to memory if you do much work with GET and PUT:

Bytes required = 4+INT ((X*bits-per-pixel+7)/8)*Y

Don't try to figure out the formula unless you have some aspirin handy! Briefly, X is the number of pixels across (the width of the object). Y is the vertical height of the object, also in pixels. You set *bits-per-pixel* to the number of bits necessary to define that pixel. It is always the number of bits required to specify the number of colors allowed in a certain mode. For single-color modes like SCREEN 2, *bits-per-pixel* is 1. Four-color modes should use 2. For the 16-color modes found only on the PCjr, use 4.

So, by applying these values to the formula, we find that the number of bytes required to hold a 20 × 10 shape in SCREEN 1 (320 × 200, four colors) is

= 4+INT((20*2+7)/8)*10
= 4+INT((40+7)/8)*10
= 4+INT(47/8)*10
= 4+(5)*10
= 54 bytes.

Now, this is the number of bytes required. But how many array elements are required when we DIM the array? Each element of an integer array requires two bytes, so we just divide the number of bytes by two:

DIM D%(27)

Instead of adding a % to the array name, you could use the statement DEFINT A–Z at the beginning of your program. This makes all variables integers unless otherwise specified. Integer FOR-NEXT loops are faster, too.

Pick Up, Put Down

Now that you've grabbed the shape with GET, you can put it down anywhere on the screen with PUT. The PUT statement takes the form of PUT (x,y),*arrayname*. To display the object we previously picked up, you can use PUT (50,50),D%. This will put a copy of the object at screen coordinates 50,50 (the coordinates always correspond to the current graphics mode).

PUT is like a rubber stamp. You create the stamp with GET, then stamp the shape down anywhere you like with PUT. PUT has several variations, however.

The most straightforward is PSET: PUT (50,50),D%,PSET. This stamps the object on top of anything already on the screen. Any blank areas in the shape erase what was beneath them. The background will not "show through" these blank areas.

Another variation is PUT (50,50),D%,PRESET. This stamps a reversed-color image of the shape (the bits are inverted from 0 to 1, and vice versa).

The statement PUT (50,50),D%,OR seems to slide the shape underneath whatever is already on the screen. In other words, your shape shows up only where blank areas existed before. There can be some color distortion where an existing image overlaps the PUT image, because the OR is done a bit at a time. Blank areas in the shape will not erase whatever they overlay.

PUT (50,50),D%,AND is an even stranger variation. If you understand binary ANDing, it's simple. Only the areas of the shape that overlay existing pixels show up on the screen. This can also cause color distortion.

The Most Exclusive PUT

The most useful variation of PUT is also the default action: PUT (50,50),D%,XOR is the same as PUT (50,50),D%. XOR means *Exclusive-OR*, an operation in binary arithmetic. The bits in the shape are Exclusive-ORed with the pixels already on the display. In practical terms, the shape looks as it should in blank areas. When it overlaps existing screen colors, though, the overlapping area is a composite of the two overlapping colors. The unique feature of XOR is that if you PUT the object in exactly the same place again, the shape will be removed, restoring the previous background area. It's somewhat like a sentence with a double negative that cancels itself out.

PUT with XOR is ideal for animation. You can draw/erase/redraw to animate without erasing any background on the screen. To minimize flicker, you should first remove the shape from the old position, then PUT it at the new location. The new location becomes the old position when the loop is continued. You can also prevent flickering by erasing and redrawing only when the position changes. The first time through the loop, skip the erase routine in order to PUT the object on the screen to begin with.

Let's try a small animation program, line by line:

100 KEY OFF:SCREEN 1:CLS

(Turn off keys, enter the 320 × 200 four-color mode, and clear screen.)

110 LINE (0,0)–(19,9),2,BF

(Draw a solid box in color 2 [purple] which is 20 units across and 10 units high.)

120 DIM D%(27)

(DIMension an integer array to hold the box when we GET it.)

130 GET (0,0)–(19,9),D%

(Grab the solid box and store it into the array D%. The animation loop follows.)

140 FOR I=1 TO 190

(Start the loop with 1 because the original box is still at position 0,0. Erase it with the first PUT on line 150.)

150 PUT (I–1,I–1),D%
160 PUT (I,I),D%

(Line 160 puts the shape at the new location. The shape moves diagonally from position 1,1 to 190,190. It must stop at 190 because that's the last legal position. Since the coordinates specify the upper-left corner of the shape, and since the shape is ten pixels long (0–9), the lower-right corner will be at position 199,199, the last legal position in this graphics mode.)

170 NEXT

(Wrap up the loop. When we exit the FOR-NEXT loop, the shape is still on the screen. If we want to remove it, we just PUT with XOR on top of the shape. Then restore screen.)

180 PUT (190,190),D%:SCREEN 0:KEY ON

Animation Variations

XOR also lets you move a shape more quickly. Just change these lines to make the shape move twice as fast by skipping every other position:

```
140 FOR I=2 TO 190 STEP 2
150 PUT (I-2,I-2),D%
```

You can even move randomly—as long as you save the position of the old shape so that you can remove it before updating the position. Study this program:

```
 10 SCREEN 1:CLS
 20 CIRCLE (5,4),5
 30 DIM CIRC%(17)
 40 GET (0,0)-(11,9),CIRC%
 50 CLS
 60 GOTO 80
 70 PUT (OLDX,OLDY),CIRC%
 80 X=INT(300*RND):Y=INT(150*RND)
 90 PUT (X,Y),CIRC%
100 OLDX=X:OLDY=Y
110 GOTO 70
```

Try this substitution for line 80:

```
80 X=STICK(0):Y=STICK(1)
```

Now you can move the shape with the joystick. The beginning of a game!

With this background, you're ready to make full use of "Sculpt-a-Shape," a graphics editor for shape design, the next article in this book. The Sculpt-a-Shape article also covers other animation techniques and gives tips for smooth, high-speed animation in BASIC.

Sculpt-a-Shape Graphics Editor for BASIC Animation

Charles Brannon

After you've learned about animating in BASIC with GET and PUT (see previous article), you may still be put off by the work needed to create detailed shapes. Now there's a utility program that takes the drudgery out of graphics work, freeing you to think artistically: "Sculpt-a-Shape." We're presenting two versions: a 4-color editor for the IBM PC with color/graphics adapter, BASICA, and at least 64K; and a 16-color version for the Enhanced Model PCjr with Cartridge BASIC.

In IBM Microsoft BASIC, the GET command is used to define and "grab" a rectangular section of the screen. Once the shape is defined and stored in an array, the PUT command can stamp the shape down anywhere on the screen. PUT has several variations that permit fairly smooth animation and special effects.

The problem with this technique is that you must first draw the shape on the screen before you can GET it. You can use the various graphics commands (such as LINE, CIRCLE, PSET, and DRAW) to sketch your design on the screen, then GET the shape, but this approach is tedious and mathematical.

"Sculpt-a-Shape" is a utility that lets you concentrate on the art. You draw the shape with the cursor keys, grab the shape, then save it as a binary file or BASIC DATA statements, ready to be MERGEd with your program. Sculpt-a-Shape lets you bypass GET altogether when defining your shapes.

Two Different Versions

Slightly different versions of Sculpt-a-Shape are designed to make the most of each computer's graphics modes. Program 1 lets you create shapes for SCREEN 1, a 4-color 320 × 200 graphics mode. It runs on either the PC or PCjr. Program 2 is a listing of line changes for Program 1. It transforms Program 1

into a shape editor for SCREEN 5, the PCjr's special 16-color 320 × 200 mode. It can also be used to create shapes for SCREEN 3, the PCjr's 16-color 160 × 200 graphics mode.

If you have a PCjr, type in and save each program separately. Save Program 2 to disk with SAVE "filename",A. That way you can write PC-compatible programs for SCREEN 1 by using Program 1. Whenever you want to create shapes for SCREEN 3 or SCREEN 5, load Program 1 and MERGE Program 2 to change the appropriate lines. You may want to save this version as well.

When you run either version of Sculpt-a-Shape, you'll see a large grid consisting of 32 blocks horizontally and 23 blocks vertically. A cursor blinks in the upper-left corner of the grid. The cursor keys move the cursor anywhere inside the grid. The grid is an eightfold enlargement of a high-resolution shape. You can draw shapes smaller than the entire grid, but no larger.

In the upper-right corner of the screen is a small box. This is where you'll see the shape in its actual size as you draw it on the large grid. Below it is another box containing a menu of editing options. In the 16-color version, you'll see a rainbow bar at the bottom of the screen, labeled with the keys you press to select a color.

Creating the Shape

Sculpt-a-Shape has two drawing modes. The first mode lets you plot each pixel (grid square) one at a time. In the 4-color version of Sculpt-a-Shape, you press the number keys 0, 1, 2, and 3 to plot a color at the cursor position. In the 16-color version, the rainbow bar shows the keys you need to press for each color. For colors 0–15, you would press 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,), !, @, #, $, or % to plot one of the 16 colors at the cursor position.

In both versions, you press 0 to erase a point (since 0 is black, the background color). Pixels are plotted only when you press one of these color keys. The cursor moves nondestructively over previously plotted points. If you want, you can draw your shape entirely in this drawing mode.

It can be laborious, though, to fill in large areas or create large shapes one point at a time. So Sculpt-a-Shape has another mode, aptly named Trail, which leaves a trail of the current color as you move the cursor. For example, in Trail mode,

you can draw a line by just moving the cursor across the grid. Remember that in Trail mode, the cursor does not move nondestructively—it fills in whatever grid points it crosses.

To enter Trail mode, press the T key. When you want to move the cursor nondestructively, press M (for Move). An indicator above the command menu shows the current mode, Move or Trail.

It's best to start drawing your shape in the middle of the grid to give yourself plenty of room to expand. Smaller shapes move faster, so for animation purposes you won't always want to create a full 32 × 23 shape.

Grabbing the Shape

After drawing your object, you have to define a rectangular portion of the grid before Sculpt-a-Shape can generate the DATA statements or binary file.

Visualize a rectangle that would frame the shape you've drawn, then place the cursor in one of the corners of the imaginary rectangle. To start defining the shape, press G (for GET or Grab). Now move the cursor to the diagonally opposite corner of the shape. A dotted-line rectangle expands from the original corner and follows the cursor as you move it. During this process, only the cursor keys are active. You can't draw or edit the shape while grabbing it.

When the dotted rectangle completely frames the shape, press G again to complete the Grab. The shape is now defined, and it appears in its actual size below the command menu.

Sculpt-a-Shape uses the GET command to grab a rectangular portion of the shape off the screen. It actually GETs from the real-size image above the command menu, not the enlarged image on the grid. That's why every time you set a point on the grid, a corresponding pixel is turned on in the box above the command menu. When a shape is grabbed, the screen bytes are transferred into an integer array. The array can be saved to disk as a binary file—ready to be called into your program with the BLOAD command—or turned into DATA statements which can be merged into a program.

It's important to distinguish between the box at the top of the screen and the actual shape at the bottom. The small box in the upper-right corner shows the entire 32 × 23 grid, but the actual shape will not be defined until you use the G command.

Action!

Once you've defined the shape, you may want to see what it looks like in motion. Press A, and the shape moves left to right across the bottom of the screen. You can observe that larger shapes move more slowly and flicker more noticeably. The animation is done with the XOR option of PUT, so the background is not erased as the shape travels.

To save the shape on disk as a binary file, press S. After you type a filename, Sculpt-a-Shape BSAVEs the contents of the array with the statement

BSAVE FILE$,VARPTR(OBJECT(0)),ENDRANGE

VARPTR points to the first memory location used by the integer array OBJECT. ENDRANGE is set to the number of memory locations (bytes) needed to define the shape. In your own program, you can BLOAD the shape directly into an array. Just make sure you reserve enough space with DIM to hold your shape. For 4-color shapes, an array size of 94 is sufficient. Use 186 for a 16-color shape. Program 3 is an example program that BLOADs a shape, then moves it around. Program 4 does the same with 16-color shapes.

You can also save the array contents with option D for DATA statements. Instead of saving the array memory directly, the Data option creates a series of DATA statements on disk. The DATA can then be read by your program into an array with a FOR-NEXT loop. You can then stamp the image anywhere on the screen with PUT, bypassing the need to GET it first. The first two bytes in the DATA statements are the X and Y size of the shape. When you apply the following formula:

E=(4+INT((X+7)/8)*Y)/2

you know how much to DIM the array, and how many DATA items to read. Program 5 is an example program that reads a shape from DATA statements into an array, then moves the shape left to right with PUT. Program 6 works in the 16-color modes, SCREEN 3 and 5. To use these programs, type them in and MERGE them with some DATA statements created with Sculpt-a-Shape.

Shapes saved as binary files can be reloaded into Sculpt-a-Shape for further editing, but you can't reload a shape stored as DATA statements. To distinguish between the two kinds of files, you might want to use the extenders .BIN for bi-

nary saves and .MRG or .DAT for DATA statements. To load a shape, just press L and give the filename. Any shape that was already on the screen is erased, and the new shape is loaded in and automatically centered on the grid. The shape is predefined, so you don't need to GET it before making DATA statements.

If you try to use Sculpt-a-Shape's Action, Save, or Data commands before you've defined the shape with GET, Sculpt-a-Shape responds with an error message: "Shape not defined. Press Enter."

The remaining command options are Color, Quit, and Rerun. In the 4-color version, press C to switch between the two palettes available in SCREEN 1. The C command cycles through the background color on the 16-color PCjr version. To clear the grid or restart the program, press R. When you are finished with Sculpt-a-Shape, Q cleanly exits back to BASIC.

A Few Guidelines

Let's say you're writing a game. Using Sculpt-a-Shape, you can design a spaceship. You GET the shape, then save it to disk as a binary file (in case you need to edit it later). You also create DATA statements on disk with the D option.

Your game is partially written, so you MERGE the DATA statements with the program. Using part of Program 5 or 6, your program reads the DATA statements for the spaceship into an integer array (you must use an integer array). Now you can enter the graphics mode (SCREEN 1, 3, or 5) and use PUT to animate the spaceship. An alternative would be to BLOAD the spaceship directly into the array. DATA statements make the program easier to share with friends, since there isn't an extra binary file to copy. (They also make the program publishable in magazines and on electronic bulletin boards.)

You can also use Sculpt-a-Shape to create several nonoverlapping images, all within the same grid. Each shape can then be grabbed and saved separately. If you run out of room on the grid, save the shape and reload it. This will center the shape within the grid, possibly giving you extra space around it.

It's not really practical to move shapes bigger than the 32 × 23 grid, since large shapes move very slowly. For fast action, try to draw small, detailed shapes. When you define the image, try to frame it exactly, wasting no space within the rectangle.

Program 1. Four-Color Version (PC or PCjr)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
QM 10 'Sculpt-a-Shape!
JF 20 'Requires BASICA or Cartridge BASIC, at lea
      st 64K, color graphics
JP 30 '
CI 100 SCREEN 1:COLOR 0,1:KEY OFF:CLS:DEF SEG
HO 110 DEFINT A-Z
CG 120 DIM PID(10),OBJECT(188):LINE (1,1)-(7,7),1
       ,B
FB 125 GET(0,0)-(7,7),PID:CLS
OI 130 FOR I=0 TO 256 STEP 8:LINE (I,0)-(I,184):N
       EXT
FO 140 FOR I=0 TO 184 STEP 8:LINE (0,I)-(256,I):N
       EXT
GP 145 LINE (270,0)-(305,26),3,B,&HAAAA:LINE (270
       ,0)-(305,26),1,B,&HCCCC:LINE (270,0)-(305,
       26),2,B,&H1111
CK 150 LOCATE 25,13:PRINT"Sculpt-a-Shape!";:LOCAT
       E 5,34:PRINT CHR$(16);"MOVE"
HM 151 ROW=9
MN 152 READ A$:IF A$="X" THEN 155
PJ 153 LOCATE ROW,34:PRINT A$:ROW=ROW+1:GOTO 152
ME 155 LINE (262,62)-(319,ROW*8-7),3,B
BH 160 X=0:Y=0:A$="0":GOTO 235
NL 170 F=0
FG 175 PUT(X*8,Y*8),PID:F=1-F
QN 180 A$=INKEY$:IF A$="" THEN 175
OB 190 IF F=1 THEN PUT(X*8,Y*8),PID
CN 195 IF A$>="a" AND A$<="z" THEN A$=CHR$(ASC(A$
       )-32)
CB 197 IF A$="R" THEN RUN
LI 198 IF A$="Q" THEN SCREEN 0,0,0:END
PJ 199 IF A$="C" THEN PAL=1-PAL:COLOR 0,1-PAL:GOT
       O 170
IN 200 IF A$=CHR$(0)+CHR$(72) THEN Y=Y+(Y>0)
OK 210 IF A$=CHR$(0)+CHR$(80) THEN Y=Y-(Y<22)
GC 220 IF A$=CHR$(0)+CHR$(75) THEN X=X+(X>0)
BK 230 IF A$=CHR$(0)+CHR$(77) THEN X=X-(X<31)
EE 231 IF A$<>"G" THEN 235
HB 232 GMODE=1-GMODE:IF GMODE THEN SX=X:SY=Y:OX=X
       :OY=Y:GOTO 236
KM 233 LINE(SX*8-L5,SY*8-L6)-(X*8+L7,Y*8+L8),3,B:
       GET (272+SX,SY+2)-(272+X,Y+2),OBJECT
QL 234 LINE (272,160)-(304,184),0,BF:PUT (272,160
       ),OBJECT,PSET
IK 235 IF GMODE=0 THEN 240
KI 236 L1=8*(OX<SX):L2=8*(OY<SY):L3=8+L1:L4=8+L2:
       L5=8*(X<SX):L6=8*(Y<SY):L7=8+L5:L8=8+L6
```

```
OH 237 LINE (SX*8-L1,SY*8-L2)-(OX*8+L3,OY*8+L4),3
       ,B:LINE (SX*8-L5,SY*8-L6)-(X*8+L7,Y*8+L8),
       2,B,&HCCCC:OX=X:OY=Y:GOTO 170
PG 240 IF A$>="0" AND A$<="3" THEN C=VAL(A$):LOCA
       TE 6,34:PRINT"COLOR":LINE (264,48)-(304,53
       ),C,BF:LINE (264,48)-(304,53),3,B:GOTO 270
EE 250 IF A$="T" THEN DRAWMODE=1:LOCATE 5,35:PRIN
       T"TRAIL"
FM 260 IF A$="M" THEN DRAWMODE=0:LOCATE 5,35:PRIN
       T"MOVE ":GOTO 170
ND 265 IF DRAWMODE=0 THEN 290
NH 270 LINE(X*8+1,Y*8+1)-(X*8+7,Y*8+7),C,BF'LINE(
       X*8,Y*8)-(X*8+8,Y*8+8),3,B
EE 280 PSET(272+X,Y+2),C
JM 290 IF A$<>"A" THEN 311
JO 291 YPOS=199-OBJECT(1)
DE 295 IF OBJECT(0)+OBJECT(1)=0 THEN 7000
BL 300 PUT (0,YPOS),OBJECT:FOR I=1 TO 319-OBJECT(
       0)/2:PUT (I-1,YPOS),OBJECT:PUT (I,YPOS),OB
       JECT:NEXT
MF 310 PUT(I-1,YPOS),OBJECT:GOTO 170
HJ 311 IF A$<>"S" THEN 320
CM 312 IF OBJECT(0)+OBJECT(1)=0 THEN 7000
JM 313 MSG$="Save image.  Filename:":GOSUB 6000:L
       INE INPUT ;FILE$:IF FILE$="" THEN 460
IH 314 ENDRANGE=4+INT((OBJECT(0)+7)/8)*OBJECT(1)
MN 315 ON ERROR GOTO 465
JG 316 BSAVE FILE$,VARPTR(OBJECT(0)),ENDRANGE
GJ 317 ON ERROR GOTO 0:GOTO 460
MJ 320 IF A$<>"D" THEN 510
CH 325 IF OBJECT(0)+OBJECT(1)=0 THEN 7000
PH 326 MSG$="Make DATA.  Filename:":GOSUB 6000:LI
       NE INPUT ;FILE$:IF FILE$="" THEN 460
AB 370 MSG$="Starting line number:":GOSUB 6000:LI
       NE INPUT ;A$:L=VAL(A$):IF A$="" THEN 460
NJ 375 ON ERROR GOTO 465
LN 380 OPEN FILE$ FOR OUTPUT AS #1
PE 390 ENDRANGE=(4+INT((OBJECT(0)+7)/8)*OBJECT(1)
       )/2
BA 400 FOR I=0 TO ENDRANGE STEP 8
MN 410 PRINT #1,L;"DATA ";:L=L+10
JA 420 FOR J=0 TO 7
OL 430 PRINT#1,"&H";HEX$(OBJECT(I+J));:IF J<7 AND
        J+I<ENDRANGE THEN PRINT#1,",";
DH 435 IF J+I=ENDRANGE THEN J=7
DG 440 NEXT:PRINT #1,"":NEXT
JO 450 CLOSE #1:ON ERROR GOTO 0
FE 460 MSG$="":GOSUB 6000:LOCATE 25,13:PRINT"Scul
       pt-a-Shape!";:GOTO 170
```

```
OF 465 MSG$="Disk Error #"+STR$(ERR)+". Press Ent
       er":GOSUB 6ØØØ:A$=INPUT$(1):RESUME 45Ø
FB 51Ø IF A$<>"L" THEN 17Ø
GO 52Ø MSG$="Load image. Filename:":GOSUB 6ØØØ:LI
       NE INPUT ;FILE$:IF FILE$="" THEN 46Ø
FH 53Ø ON ERROR GOTO 465:BLOAD FILE$,VARPTR(OBJEC
       T(Ø)):ON ERROR GOTO Ø
ED 535 XOFF=16-OBJECT(Ø)/4:YOFF=11-OBJECT(1)/2:XO
       FF=-XOFF*(XOFF>Ø):YOFF=-YOFF*(YOFF>Ø)
KF 54Ø LINE(Ø,Ø)-(256,184),Ø,BF:LINE(271,1)-(3Ø4,
       25),Ø,BF:PUT(272+XOFF,2+YOFF),OBJECT
FG 541 FOR I=Ø TO 184 STEP 8:LINE (Ø,I)-(256,I):N
       EXT:FOR I=Ø TO 256 STEP 8:LINE (I,Ø)-(I,18
       4):NEXT
QK 542 LINE (272,16Ø)-(3Ø4,184),Ø,BF:PUT (272,16Ø
       ),OBJECT,PSET
PE 55Ø XF=XOFF*8:YF=YOFF*8:GOSUB 2ØØØ:GOTO 46Ø
LO 2ØØØ FOR I=Ø TO OBJECT(1):FOR J=Ø TO OBJECT(Ø)
        /2-1:LINE (XF+J*8+1,YF+I*8+1)-(XF+J*8+7,Y
        F+I*8+7),POINT(XOFF+J+272,YOFF+I+2),BF:NE
        XT:NEXT:RETURN
JI 5ØØØ DATA Ø 1 2 3,A)ction,C)olor,D)ata,G)et,L)
        oad,M)ove,Q)uit,R)e-run,S)ave,T)rail,X
HP 6ØØØ LINE (Ø,192)-(319,199),Ø,BF:LOCATE 25,1:P
        RINT MSG$;:RETURN
MJ 7ØØØ MSG$="Shape not defined. Press Enter":BEE
        P:GOSUB 6ØØØ:A$=INPUT$(1):GOTO 46Ø
```

Program 2. Modifications to Program 1 (PCjr Only)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
GF 2Ø 'Modifications for PCjr Cartridge BASIC--16
      colors (SCREEN 5)
DD 4Ø CLEAR ,,,32768!
NF 5Ø C$="Ø123456789)!@#$%"
MC 1ØØ SCREEN 5:KEY OFF:CLS:DEF SEG
EI 12Ø DIM PID(36),OBJECT(372):LINE (1,1)-(7,7),1
       5,B
QJ 15Ø LOCATE 25,2Ø:PRINT"Sculpt-a-Shape!";:LOCAT
       E 5,34:PRINT CHR$(16);"MOVE"
MD 155 LINE (262,62)-(319,ROW*8-7),,B
OK 157 FOR I=Ø TO 15:LINE (I*8,185)-(I*8+7,19Ø),I
       ,BF:NEXT:LOCATE 25,1:PRINT C$;
IE 199 IF A$="C" THEN BK=(BK+1) AND 15:COLOR ,BK:
       GOTO 17Ø
BB 232 GMODE=1-GMODE:IF GMODE THEN SX=X:SY=Y:OX=X
       :OY=Y:GOTO 236:ELSE IF SX=X THEN BEEP:GMOD
       E=1:GOTO 236
```

```
BL 233 LINE(SX*8-L5,SY*8-L6)-(X*8+L7,Y*8+L8),,B:G
       ET (272+SX,SY+2)-(272+X,Y+2),OBJECT
NB 237 LINE (SX*8-L1,SY*8-L2)-(OX*8+L3,OY*8+L4),,
       B:LINE (SX*8-L5,SY*8-L6)-(X*8+L7,Y*8+L8),2
       ,B,&HCCCC:OX=X:OY=Y:GOTO 170
HJ 240 CC=INSTR(C$,A$):IF CC THEN C=CC-1:LOCATE 6
       ,34:PRINT "COLOR":LINE(264,48)-(304,53),C,
       BF:LINE (264,48)-(304,53),,B:GOTO 270
BF 460 MSG$="":GOSUB 6000:LOCATE 25,20:PRINT"Scul
       pt-a-Shape!";
KA 461 FOR I=0 TO 15:LINE (I*8,185)-(I*8+7,190),I
       ,BF:NEXT:LOCATE 25,1:PRINT C$;:GOTO 170
KP 535 XOFF=16-OBJECT(0)/8:YOFF=11-OBJECT(1)/2:XO
       FF=-XOFF*(XOFF>0):YOFF=-YOFF*(YOFF>0)
BO 2000 FOR I=0 TO OBJECT(1):FOR J=0 TO OBJECT(0)
        /4-1:LINE (XF+J*8+1,YF+I*8+1)-(XF+J*8+7,Y
        F+I*8+7),POINT(XOFF+J+272,YOFF+I+2),BF:NE
        XT:NEXT:RETURN
GK 5000 DATA A)ction,C)olor,D)ata,G)et,L)oad,M)ov
        e,Q)uit,R)e-run,S)ave,T)rail,X
```

Program 3. BLOADer and Animator (4-Color Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
CK 10 'Bloader for shape files
CI 100 DEFINT A-Z:DIM OBJECT(94):SCREEN 1:CLS
EC 110 BLOAD "filename",VARPTR(OBJECT(0))
JF 120 PUT(0,0),OBJECT:FOR I=1 TO 319-OBJECT(0):P
       UT(I-1,0),OBJECT:PUT(I,0),OBJECT:NEXT
```

Program 4. BLOADer and Animator (16-Color Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
CK 10 'Bloader for shape files
GG 100 CLEAR,,,32768!:DEFINT A-Z:DIM OBJECT(186):
       SCREEN 5:CLS
EC 110 BLOAD "filename",VARPTR(OBJECT(0))
JF 120 PUT(0,0),OBJECT:FOR I=1 TO 319-OBJECT(0):P
       UT(I-1,0),OBJECT:PUT(I,0),OBJECT:NEXT
```

Program 5. DATA Reader and Animator (4-Color Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
AE 10 DEFINT A-Z:SCREEN 1:CLS
PM 20 READ X,Y:E=(4+INT((X+7)/8)*Y)/2:DIM O(E):O(
      0)=X:O(1)=Y:FOR I=2 TO E:READ O(I):NEXT
OG 30 PUT (0,0),O
NE 40 FOR I=1 TO 319-O(0):PUT(I-1,0),O:PUT(I,0),O
      :NEXT
```

Program 6. DATA Reader and Animator (16-Color Version)

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KD 1Ø CLEAR,,,32768!:DEFINT A-Z:SCREEN 5:CLS
PM 2Ø READ X,Y:E=(4+INT((X+7)/8)*Y)/2:DIM O(E):O(
      Ø)=X:O(1)=Y:FOR I=2 TO E:READ O(I):NEXT
OG 3Ø PUT (Ø,Ø),O
NE 4Ø FOR I=1 TO 319-O(Ø):PUT(I-1,Ø),O:PUT(I,Ø),O
      :NEXT
```

Monochrome Graphics

Michael A. Covington

Some graphics screens don't require a color/graphics adapter. You can create graphics on any IBM PC or PCjr, even those equipped with only the monochrome display board.

Owners of the IBM Monochrome Display often feel a bit left out because their systems don't support point-by-point graphics. The Monochrome Display Adapter can produce a variety of special characters that could be used to construct drawings, but most programs never employ more than a few of them.

The brief program accompanying this article takes advantage of these special characters to create dramatic-looking patterns—actually contour maps of three-dimensional mathematical functions. Some of the displays look rather like Scottish tartans. The program runs on a PC or PCjr with any display, but the IBM Monochrome Display yields the best results.

Line 160 in the program defines the variable W as a function of ROW and COL. The function can be anything you wish. Here are some functions that result in attractive patterns:

W = ROW+COL
W = ROW*COL
W = LOG(ROW)−LOG(COL)
W = LOG(ROW^2+COL^2)
W = 5*SIN((ROW+COL)/10)

An almost infinite variety of other functions are possible. Just substitute your function for line 160, save, and then run the program.

Monochrome Graphics

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
EO 10 WIDTH 80
BP 30 SCRNWIDTH = 80
CE 50 DIM A$(4)
KH 60 A$(0)=" "
MM 70 A$(1)=CHR$(176)
NF 80 A$(2)=CHR$(177)
```

```
PO 90 A$(3)=CHR$(178)
HC 100 A$(4)=CHR$(219)
HI 110 CLS:KEY OFF
CP 120 FOR ROW=1 TO 22
JM 130 FOR COL=1 TO SCRNWIDTH-1
NP 140 ' Change the following line
EK 150 ' to get different patterns
OH 160 W = SQR((2*(ROW-11))^2 + (COL-40)^2)
AF 170 PRINT A$(ABS(W) MOD 5);
ON 180 NEXT
JC 190 PRINT
NO 200 NEXT
PP 210 BEEP
```

CHAPTER 6

Programming

BASIC Hints

C. Regena

The BASIC in your PC or PCjr is powerful, but learning to fully use all the commands can be difficult and time-consuming. Here are some hints and tips to assist you.

Among the many letters I get are numerous questions about BASIC on the IBM. In this article I've put together some hints and tips, discuss the IF-THEN statement, and give you a sample program.

Programming Hints and Tips

All BASIC lines must be numbered. Line numbers may be from 0 to 65529.

The command AUTO makes the computer automatically type the line numbers for you as you're programming. AUTO will start the automatic numbering with the current line number and increment by tens. Press Fn-Break to disable automatic numbering. Do not edit while in AUTO mode.

RENUM is the command to renumber or resequence your program. All line numbers referenced in other statements will automatically be renumbered.

LIST . will list the last line you typed in, edited, or LISTed.

Use REM to indicate a REMark statement. Abbreviate this command by typing the single quote mark, or apostrophe:

```
10 ' TITLE
```

You can separate commands on a line with the colon, but if the last statement is a REMark, the colon and REM may be replaced by the apostrophe:

```
70 SC=SC+1 ' INCREMENT SCORE
```

Variable Names

Variable names may be up to 40 characters long. Actually, they may be longer, but only the first 40 characters will be recognized. (Several microcomputers allow only two characters

in a variable name; others permit unlimited length.) The variable name may contain embedded reserved BASIC words without any problems. Some computers will not accept such names as COST, because it contains COS, or SCORE, because it contains OR.

Fn-Pause stops a program listing or freezes a running program. Press any key to continue.

Fn-Break also stops a listing or a program, but it prints a message. To continue, type CONT and press Enter.

When you use a function key, press Fn first, then the second key. However, when you use Alt or Ctrl, it must be held down as you press the second key.

Ctrl-Alt-Del (the Control key at the left of the keyboard, the blue Alt key, and the Delete key) pressed simultaneously reset the computer system. The result is just as if you turned the computer off and then on again.

Press Ctrl-Alt and either the left-arrow key or right-arrow key to move the screen image left or right on the monitor or television. With some televisions you need to center your screen to read all the columns.

For inequality you may use either <> or ><.

Arrays

An array may have 235 dimensions. The maximum number of elements per dimension is 32767.

If you use a subscripted number without a previous DIMension statement, the computer assumes DIM X(10), or the 11 elements from 0 to 10. A regular variable may have the same name as an array variable without causing any problems. For example, you can use both A$ and A$(3,5) in the same program. They are treated as separate variable names.

Filenames, such as names of programs you save, may be up to eight characters long. Disk filenames may be eight characters long plus a period and three-character extension.

To use the disk drive and program in BASIC on the PCjr, you must have Cartridge BASIC. Before you can save a program, the disk must be *initialized,* or *formatted.* Turn on the computer with the DOS disk in the drive. The FORMAT command is part of DOS.

FORMAT has several options. FORMAT /S formats the disk and transfers the Disk Operating System commands to your program disk so that you don't need to insert the DOS

disk when returning to DOS. The /V option when formatting allows you to put a title on a disk (*V*olume label). After the computer formats the disk, it prompts you for a title, which may be up to 11 characters long. Type FORMAT /S/V to use both the system option and the volume label option.

To save a program in protected form, use P after the filename. This protection keeps someone from LISTing, editing, or saving the program. This is available in Cassette BASIC or Cartridge BASIC, on cassette or disk. An example is SAVE "MYGAME",P.

Automatic Load and Run

Following is a procedure to automatically load and run a program after you turn on the computer:

1. Type in or load your BASIC program. For an example, try this program.

```
10 REM PRINTS DIRECTORY
20 CLS
30 FILES
40 NEW
50 END
```

2. Save your program on the disk as usual.

```
SAVE "DISK"
```

3. Type SYSTEM and press Enter to return to DOS.
4. Type COPY CON: AUTOEXEC.BAT and press Enter.
5. Type BASIC DISK and press Enter—or use the name of the program you want to run automatically.
6. Press Fn-6, then press Enter. You will see a Z on the screen. The drive will whir momentarily.

Now when you insert the disk at the beginning of a session, the program you named will automatically load and run.

The sample program above will print a directory of the files on the disk. Line 40 erases the program so you can enter or load another program.

Using the Function Keys

Remember to use the function keys to save typing. To load a program, press Fn-3, then type the name of the program. You don't need to type the last quote, and the name may be in uppercase or lowercase letters.

Fn-1 is LIST, and you may specify lines or list the whole program. Fn-2 is RUN and includes Enter. Fn-3 is LOAD", and Fn-4 is SAVE". Just fill in the name of the program. Fn-5 is CONT and is used to continue running a program after you have pressed Fn-Break.

Fn-6 is ,"LPT1:" and Enter. You type in the beginning of the command for what you want on the printer. Fn-7 is TRON, and Fn-8 is TROFF, and you don't need to press Enter. These are TRACE commands for debugging programs. Fn-9 is the word KEY; add whatever you need to complete the command (such as KEY OFF). Fn-10 is SCREEN 0,0,0 and Enter, which is very handy if you're working in a graphics mode and want to clear the screen and return to the normal text screen.

If you have forgotten what files or programs are on your disk, use the DOS command DIR (for directory). The files or program names will be listed along with the size of each file and the date and time you saved it. If you are in BASIC and need to get to DOS, type SYSTEM and press Enter, then DIR for directory. If you don't want to lose your BASIC program by typing SYSTEM, type the command FILES to see the disk directory. Only the filenames are shown.

PRINT USING

Quite a few commands are available in Cassette BASIC which are not discussed in the *Hands-On BASIC* book that comes with the PCjr. The *BASIC Reference Guide* that comes with Cartridge BASIC is a necessity if you like to program.

One useful command is PRINT USING, which can help format your output. You can use commas, semicolons, and the TAB function in your regular PRINT statements to line up columns and make the screen look nice, but often PRINT USING is easier.

One of PRINT USING's handiest features is right justification of a column of numbers (lining up the decimal places). A regular PRINT statement will start printing the number in the next print position. The numbers will line up fine—as long as their lengths are the same (for example, all two-digit numbers between 15 and 70). PRINT USING will right justify the numbers to line up the ones column, the tens column, and so forth. The basic command with numbers is

PRINT USING "###";*N*

where *N* is a variable name for a number. The number signs (#) each represent a digit. This format allows for a three-digit number.

You may specify a certain number of decimal places—and the computer will round (that's round, not truncate) to that number of places. PRINT USING "##.###" will print numbers under 100 and allow up to three decimal places. If the number rounds to 100 or more, a percent sign will be printed before the number, but the number will still be printed correctly. If the number is less than 1, a zero will be printed to the left of the decimal.

If you are printing large numbers, you may want to use commas in your numbers for thousands or millions (or billions). As you know, the computer just prints the number with no commas. You could use string functions and figure out the length of the number (number of digits), then combine parts of strings to get the number with commas in the right places. But PRINT USING will automatically place commas in large numbers. Use a comma just before the decimal point, and a comma will be printed to the left of every third digit to the left of the decimal point. PRINT USING "####,.##" will place commas for the thousands.

Dollars and Cents

If you use your computer to print reports involving money, PRINT USING "$ ###.##";PRICE will print the number rounded to two decimal places and will put a dollar sign and a space before the number. By the way, specifying the two places after the decimal point makes sure that zeros will be printed, if necessary, to fill in the places. A price that would ordinarily turn out to be 256.2 will be printed $ 256.20 in this format. Two dollar signs just before the number symbols will print the dollar sign immediately before the number.

You may combine a message with the number, such as PRINT USING "THE ANSWER IS ##.##";A. You can also put a plus sign before or after the number symbols to print the sign of the number before or after the number. Two asterisks before the number symbols cause leading spaces to be filled in with asterisks. There are other options available, too, so you might take a few minutes and experiment with them to become familiar with this handy command.

IF-THEN

Conditional branching is what makes a computer seem intelligent. Actually, of course, the programmer has to tell the computer what to do. IF-THEN statements direct the computer to branch a certain way if a certain condition is true. IBM BASIC also allows ELSE, which directs a branch if the condition is not true.

The basic form is IF *expression* THEN *clause* ELSE *clause*, where *expression* is a numeric expression or a condition to be tested and *clause* may be either another BASIC command or a line number.

```
280 IF SCORE=10 THEN 370
```

tells the computer to check if the variable SCORE is equal to 10. If so, the program branches to line 370. If not, the program ignores everything after THEN and simply goes to the next line.

```
280 IF SCORE=10 THEN 370 ELSE 180
```

will branch to line 370 if the expression SCORE=10 is true and will branch to line 180 if the expression is false.

Commands also are allowed after THEN and ELSE, and there may be several commands separated by colons:

```
280 IF SCORE=10 THEN PRINT "YOU
    WIN":G=G+1:GOTO 470
```

A numeric expression which includes any of the arithmetic operators may be used. String expressions may be compared. If a condition is false, the value of the expression is 0; if it is true the value is −1. The following are acceptable expressions.

```
200 IF A THEN 220
300 IF N$<>"ZZZ"THEN GOTO 500
400 IF LEN(P$)>3 THEN PRINT P$
500 IF B/D<C*A THEN A=A+1
600 IF X-Y THEN Z=10
```

Logical operators may be used. The words accepted are NOT, AND, OR, XOR, EQV, and IMP. For example,

```
700 IF A$<"1" OR A$>"4" THEN 650
800 IF A=10 AND N$>"M" THEN PRINT N$
```

C H A P T E R 6

The Sample Program

The sample program, "Buying Items," illustrates several types of IF-THEN-ELSE statements and several examples of PRINT USING.

Buying Items is a mathematics competency program which presents problems similar to word problems in achievement tests. The problem can be written in several ways, depending on a form chosen randomly. There are six possible names that can be used in the problem. Notice that the names are all five letters long, so we know how the printing will turn out. The first three names are girls' names and the last three are boys' names. IF-THEN statements will print the proper pronoun "she" or "he" depending on the name chosen.

One of three lists is printed on the screen with randomly chosen prices, calculated from numbers read in as C1 and expressed in cents. The numbers are divided by 100, and PRINT USING prints the numbers in standard dollars-and-cents form. The first question asks how much it would cost to buy all the items on the list. The second question randomly chooses a sales tax rate and asks how much the total would be with sales tax. The third question shows how a multiple-choice answer can be set up. Although this type of problem is on a high-school math competency test, only third-grade level addition and multiplication are required.

Line Number	Explanation
20	Defines text screen with width of 40 columns; turns on the color; removes the function key labels at the bottom of the screen.
30	Sets background, border, and foreground colors; clears screen; positions cursor for printing.
40–70	Prints title of program.
80–140	Defines variables by READing values from DATA. Arrays are used, and the values are read in with nested FOR-NEXT loops.
150–160	READs in six names that can be used in the problems.
170–190	Defines H$ strings for use in the multiple-choice answers.
200	Defines color for printing and blinking.
210	Prints blinking message; plays a prompting beep.

220	Waits for user to press a key before the program goes to the next line. E=RND generates random numbers while waiting to randomize later selections.
230–240	Changes color of screen, border, and printing, and clears screen.
250	Randomly chooses one of three categories.
260	Initializes total price.
270–320	Prints list of items with randomly chosen prices; calculates total price.
330–390	Randomly chooses one of two forms for the question and prints the question.
400	Plays a prompting beep and asks for answer.
410–440	If answer is correct, plays an arpeggio; if answer is incorrect, prints instructions and correct answer.
450–470	Randomly chooses sales tax rate and prints question.
480	Asks for answer.
490	Calculates total price with sales tax, rounded.
500–520	If answer is correct, plays an arpeggio; if answer is incorrect, prints instructions and correct answer.
530	Calls subroutine to wait for user to press any key to continue.
540–580	Clears screen; reprints list of items with costs.
590–640	Prints question depending on form chosen and name chosen.
650	Randomly chooses correct answer R.
660–740	Prints four possible answers.
750–760	Receives user's answer, which must be a letter from *A* to *D*, either lowercase or uppercase.
770–800	Checks answer.
810–850	Presents option to try again and branches appropriately.
860–880	Subroutine to wait for user to press any key to continue.
890–920	Subroutine to print "Correct!" and play an arpeggio.
930	Clears screen; returns to black-and-white text screen.
940	Ends.

Buying Items

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
IG 10 REM BUYING ITEMS WITH SALES TAX
LJ 20 SCREEN 0,1:KEY OFF:WIDTH 40
CN 30 COLOR 7,1,9:CLS:LOCATE 2,1,0
QN 40 PRINT TAB(10)"*******************"
CC 50 PRINT TAB(10)"* MATH COMPETENCY *"
QP 60 PRINT TAB(10)"*******************"
BK 70 PRINT:PRINT TAB(6)"BUYING ITEMS WITH SALES
      TAX"
IC 80 FOR A=1 TO 3:FOR C=1 TO 5
FF 90 READ J$(A,C),C1(A,C,1),C1(A,C,2)
IL 100 NEXT C,A
PC 110 DATA Pencil,8,7,Eraser,2,8,Notebook,35,64,
       Ruler,29,20
IO 120 DATA Paper,59,30,Doll,249,350,Ball,49,40,T
       ruck,100,50,Game,270,230
HC 130 DATA Model,300,400,Candy,20,30,Meat,123,30
       0,Fruit,24,26
KB 140 DATA Chips,100,150,Bread,100,80
AF 150 FOR A=1 TO 6:READ N$(A):NEXT
GB 160 DATA Cindy,Chery,Chris,Ricky,Bobby,Randy
FI 170 H$(1)="Pencil and Eraser"
KO 180 H$(2)="Ball and Truck"
NC 190 H$(3)="Candy and Fruit"
PN 200 COLOR 23,1
IK 210 LOCATE 23,10:PRINT "PRESS ANY KEY TO START
       ":BEEP
EC 220 E$=INKEY$:IF E$="" THEN E=RND:GOTO 220
JN 230 COLOR 0,3,11
AG 240 CLS
MI 250 A=INT(3*RND+1)
GN 260 TP=0
KL 270 PRINT "Given this price list:":PRINT
EG 280 FOR C=1 TO 5
OH 290 P(C)=(C1(A,C,1)+INT(C1(A,C,2)*RND))/100
MB 300 PRINT TAB(4);J$(A,C),
CM 310 PRINT USING "$ ##.##";P(C)
DP 320 TP=TP+P(C):NEXT C
MD 330 F=INT(2*RND+1)
AG 340 IF F=2 THEN 370
BA 350 PRINT:PRINT "How much will it cost to buy
       all the"
PA 360 PRINT "items on the list?":GOTO 400
CP 370 N=INT(6*RND+1)
LM 380 PRINT:PRINT N$(N);" wants to buy everythin
       g on the"
HI 390 PRINT "list.  What would the total cost be
       ?"
```

```
DG 400 BEEP:INPUT X
CF 410 IF ABS(X-TP)<.001 THEN GOSUB 890:GOTO 450
PF 420 PRINT:PRINT "Add all five numbers."
JC 430 PRINT "The total is ";
AB 440 PRINT USING "$ ###.##";TP
EA 450 S=INT(7*RND+1)
PL 460 PRINT:PRINT "Sales tax is";S;"percent."
GN 470 PRINT "What is the final cost?"
NE 480 INPUT Y
MA 490 TP=TP+TP*S/100:TP=INT(100*(TP+.005))/100
FH 500 IF ABS(Y-TP)<.005 THEN GOSUB 890:GOTO 530
MI 510 PRINT:PRINT "Multiply";S;"percent times to
       tal."
KE 520 PRINT USING "The answer is $ ###.##";TP
MH 530 GOSUB 860
BJ 540 CLS
ED 550 FOR C=1 TO 5
NP 560 PRINT TAB(4);J$(A,C),
DK 570 PRINT USING "$ ##.##";P(C)
NH 580 NEXT C:PRINT
PN 590 IF F=1 THEN PRINT "If you could only";ELSE
        PRINT "If ";N$(N);" could only";
HA 600 IF A=1 THEN M=INT(5*RND+25) ELSE IF A=2 TH
       EN M=INT(36*RND+239) ELSE M=INT(18*RND+100
       )
OA 610 PRINT USING " spend $##.##";M/100
NP 620 PRINT "which of these pairs of items on th
       e"
JH 630 IF F=1 THEN PRINT "list could you buy?":GO
       TO 650
CE 640 IF N<4 THEN PRINT "list could she buy?" EL
       SE PRINT "list could he buy?"
JM 650 R=INT(4*RND+1):PRINT
OB 660 FOR V=1 TO 4
OJ 670 IF V=R THEN S$(V)=H$(A):GOTO 730
IO 680 X=INT(2*RND+4):S$(V)=J$(A,X):X=INT(3*RND+1
       )
LO 690 S$(V)=S$(V)+" and "+J$(A,X)
HE 700 IF V=1 THEN 730
ML 710 FOR V1=1 TO V-1:IF S$(V1)=S$(V) THEN 680
HO 720 NEXT V1
CD 730 PRINT CHR$(64+V);" "+S$(V)
CA 740 NEXT V:PRINT:BEEP
PH 750 E$=INKEY$:IF E$>="A" AND E$<"E" OR E$>="a"
        AND E$<"d" THEN 760 ELSE 750
OD 760 PRINT E$:E=ASC(E$)
JB 770 IF E=64+R OR E=96+R THEN GOSUB 890:GOTO 81
       0
CI 780 PRINT "The total of the two items must be"
```

```
BF 790 PRINT USING "less than $###.##";M/100
GM 800 PRINT "The answer is "CHR$(64+R)
CD 810 PRINT:PRINT "Try again? (Y/N)"
HD 820 E$=INKEY$
FK 830 IF E$="N" OR E$="n" THEN 930
IB 840 IF E$<>"Y" AND E$<>"y" THEN 820
ED 850 GOTO 240
JK 860 PRINT:PRINT "PRESS ANY KEY TO CONTINUE"
DC 870 A$=INKEY$:IF A$="" THEN 870
NC 880 RETURN
IB 890 PRINT:PRINT "CORRECT!"
FO 900 SOUND 262,2:SOUND 330,2
BG 910 SOUND 392,2:SOUND 523,4
MH 920 RETURN
BH 930 CLS:SCREEN 0,0
MK 940 END
```

Beginning BASIC: READ, DATA, and RESTORE

C. Regena

Using the BASIC statements READ, DATA, and RESTORE, you can store and manipulate information from within a program. The example program included runs on any PCjr or a PC with the color/graphics adapter.

Among all the letters I receive from people who have typed in my programs from listings in books or magazines, the most common question or problem has to do with DATA statements.

A DATA statement is always associated with a READ statement, and together they essentially perform LET or assignment processes. Omitting DATA statements can sometimes make a program easier to understand (with less chance for error), but they can combine many repetitious lines and thus save memory and make the program more efficient. By the way, the command LET is optional in IBM BASIC commands. LET A=4 may also be written A=4.

Suppose we want to initialize several variables, then print some combinations of the numbers. The program segment would be

```
10 A=4
20 B=7
30 C=3
40 D=5
50 E=12
60 F=2
70 PRINT A+B,C*D,E/F
```

Using DATA and READ, lines 10–60 may be combined like this:

```
10 READ A,B,C,D,E,F
20 DATA 4,7,3,5,12,2
70 PRINT A+B,C*D,E/F
```

When the computer comes to a READ statement, it looks for the first DATA statement. The first variable in the READ statement will correspond with the first number in the DATA statement. In this case, the computer will note that the value for A is 4. The computer will then READ B and will go to the very next DATA item to assign 7 to B. This process continues.

The DATA statement may be placed anywhere in the program. For example, you may change line 20 to line 5 and the program will work exactly the same. You may change it to line 80 if you prefer. The computer ignores DATA statements until a READ statement is encountered; then the computer will look at the DATA statements in order.

Not Just Numbers

Your data items may also be strings. Quite often you will see a READ statement in a loop to perform repeated operations, perhaps using subscripted variables or variables in an array:

```
10 FOR C=1 to 10
20 READ N$(C)
30 NEXT C
40 DATA CHERY,RICHARD,CINDY,BOB,RANDY
50 DATA ROGER,SHERYL,GRANT,DOUG,SHEILA
```

In this program segment, N$(1) will be CHERY, N$(2) will be RICHARD, and so on. Since very long lines (255 characters) are accepted on the PC and PCjr, all of the data may be typed on one line. The computer keeps track of a *data pointer* to know how much data has been used. If all the data is finished in one line, the computer goes to the next DATA statement if it needs more. If you do not have enough data items, however, the computer will print an error message.

Your job as a programmer is to make sure the data matches the READ statements and that items are read in the right order. You may combine numbers and strings in the same statements as long as you make sure the numbers go to numeric variable names and the strings go to string variable names. If you have extra data items, the computer simply ignores them.

The RESTORE statement lets you reuse the data items or makes sure the computer starts with the very first DATA statement in the program. RESTORE moves the data pointer from whatever data items have already been used back to the first item. Here is an example:

```
 10 FOR I=1 TO 3
 20 READ A,B
 30 PRINT A;"+";B;"=";A+B
 40 NEXT I
 50 PRINT
 60 RESTORE
 70 FOR I=1 TO 2
 80 READ X,Y
 90 PRINT X;"*";Y;"+";X*Y
100 NEXT I
110 DATA 2,4,8,5,7,3
```

When you run this program, A and B will first be 2 and 4, then 8 and 5, then 7 and 3. Line 60 RESTOREs the data. X and Y will then be read as 2 and 4 the first time through the loop, and 8 and 5 the second time through the loop. We didn't need to use all the data. If there was another READ statement later in the program, the data value would be 7.

The RESTORE statement starts the DATA over with the very next READ statement. In the program above, it wouldn't matter if we interchanged lines 50 and 60; the result would be the same. The RESTORE and READ statements may be separated by several statements. Notice, however, that the program would be different if you put the RESTORE statement at line 75.

If you'll spend a little time experimenting with DATA, READ, and RESTORE, you'll soon understand how they work. Try putting your DATA statements in different places in the program. Try using different numbers of items in the DATA statements. In the names example above, we could have all ten names in one DATA statement, or we could have each name in a different DATA statement.

More About RESTORE

A very useful feature of IBM BASIC is that we can specify a line number with the RESTORE command. RESTORE 800 in a program means the next READ statement will start with the DATA in line 800. This command can really help you keep track of DATA statements. If you have a long program with a lot of data, you can arrange your data properly, then use RESTORE *xxx* before each READ statement so you know exactly which data goes with which segment of the program.

C H A P T E R 6

Southern States

"Southern States" is a drill-and-practice type of program in which the outline of a state is shown. The user must type the name of the state correctly. If the state is typed correctly, the program asks for the capital city. This quiz is for the southern part of the United States, and it selects the 11 states randomly. If an answer is incorrect twice, the correct answer is shown, and that state will appear again later in the quiz. Once a state and capital have been answered correctly, they will not reappear. The quiz is over when all 11 states and capitals have been correctly named.

The program draws the map on the medium-resolution screen, SCREEN 1, using the LINE command. If you have Cartridge BASIC, the DRAW command is much simpler to use. This program, however, can be used with Cassette BASIC as it is.

You may use the ideas in this program to develop different sections of the United States (or the complete set of states), the provinces of Canada, the countries of a certain continent, the continents of the world, or the counties in a state.

The 11 states and capitals are read as data in lines 130–170. The states are S$, and the capitals are C$. Since there are 11 states, we need a DIMension statement, line 20. I could have used subscripts 0 to 10, but I think it's easier to use 1 to 11. Notice that line 130 uses a FOR-NEXT loop to read the names. The data in lines 140–170 must be arranged in the proper order. First is a state name, then its capital. S$(1) will be Texas, and C$(1) will be Austin. S$(2) will be Oklahoma, and C$(2) will be Oklahoma City, and so on. I don't use a RESTORE here because these are the first READ statements the program encounters and also the first DATA statements.

Line 240 says to RESTORE 250. This means the next READ statement starts with the data in line 250. Lines 250–350 contain data which is used to draw the map. Line 370 tells the computer to READ X and Y 97 times, then line 380 draws a line from the last point to the new (X,Y). Please be careful typing the numbers. These are coordinates for the LINE command. These lines of data could be much longer—more data per line number—but to make the typing easier I kept all the lines to two screen lines.

After the map is drawn, the computer chooses a random number, R, from 1 to 11. Line 420 uses ON R GOSUB to go to different subroutines depending on the value of R. The subroutines are in lines 780 to 1150. Each subroutine RESTOREs a certain line of data for that state, then RETURNs. Line 430 READs the first four numbers in the proper DATA statement and uses those four numbers in the LINE command. Line 440 reads the next number in the DATA statement, C, which will then be used as a limit in the FOR-NEXT loop. This number C tells how many more lines must be drawn to outline the state. Lines 450–470 then READ X and Y coordinates to draw the lines, so the rest of the data items in the DATA statements of the subroutines are coordinate numbers. I left the DATA statements with the different subroutines so that you could see which data goes with which state.

Southern States

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
BF 10 REM SOUTHERN STATES
JL 20 DIM S$(11),C$(11)
EF 30 KEY OFF:CLS:SCREEN 0:WIDTH 40:DEF SEG=0:POK
      E 1047,64
EH 40 LOCATE 2,10,0:PRINT "** SOUTHERN STATES **"
DF 50 LOCATE 5,3:PRINT "You will see an outline o
      f a state."
OC 60 LOCATE 7,3:PRINT "Type the name of the stat
      e,"
HB 70 LOCATE 9,3:PRINT "then its capital city."
LK 80 LOCATE 12,3:PRINT "If you get the state and
       capital"
MM 90 LOCATE 14,3:PRINT "correct, it will not app
      ear again."
NF 100 LOCATE 17,3:PRINT "The quiz consists of 11
       "
DC 110 LOCATE 19,3:PRINT "southern states chosen
       randomly."
FA 120 REM READ IN STATES
IN 130 FOR C=1 TO 11:READ S$(C),C$(C):NEXT C
NE 140 DATA TEXAS,AUSTIN,OKLAHOMA,OKLAHOMA CITY,A
       RKANSAS,LITTLE ROCK
EM 150 DATA LOUISIANA,BATON ROUGE,MISSISSIPPI,JAC
       KSON,TENNESSEE,NASHVILLE
CP 160 DATA ALABAMA,MONTGOMERY,GEORGIA,ATLANTA,FL
       ORIDA,TALLAHASSEE
DF 170 DATA NORTH CAROLINA,RALEIGH,SOUTH CAROLINA
       ,COLUMBIA
```

```
PC 180 LOCATE 23,7:PRINT "<Press any key to start
       >":BEEP
ME 190 A$=INKEY$:IF A$="" THEN A=RND(1):GOTO 190
IL 200 REM QUIZ
KM 210 FOR N=1 TO 11
MC 220 F=0
HI 230 CLS:SCREEN 1:COLOR 10,0
IJ 240 RESTORE 250:DATA FOR MAP
LH 250 DATA 140,32,178,32,178,28,182,28,180,32,19
       9,35,190,42,195,42,205,37
GM 260 DATA 203,40,210,43,216,40,222,43,218,43,22
       4,46,223,53,224,54,227,51
NG 270 DATA 230,56,228,64,233,63,243,54,243,53,25
       1,48,256,40,268,36,270,28
IO 280 DATA 271,22,273,23,277,21,283,31,284,30,28
       4,33,275,40,274,46,277,51
FI 290 DATA 279,51,279,49,279,51,277,51,264,61,26
       6,63,263,79,266,85,264,91
IM 300 DATA 246,110,246,118,256,135,256,142,254,1
       45,248,140,243,133,242,126
LK 310 DATA 236,123,230,125,227,123,210,126,210,1
       29,212,131,202,131,202,129
CO 320 DATA 198,128,196,130,190,130,172,138,172,1
       47,175,149,166,148,162,144
HH 330 DATA 163,142,153,130,148,130,144,133,139,1
       30,136,123,134,123,130,119
GL 340 DATA 123,118,123,120,108,118,93,110,93,108
       ,83,108,82,104,79,101,70,95
PM 350 DATA 64,80,67,76,64,76,62,72,60,62,63,54,6
       4,50,71,35,72,24,78,26,80,23
PN 360 LINE (80,23)-(112,28)
PG 370 FOR T=1 TO 97:READ X,Y
HK 380 LINE -(X,Y)
FJ 390 NEXT T
AC 400 REM DRAW STATE
NE 410 R=INT(11*RND+1):IF S$(R)="" THEN 410
ND 420 ON R GOSUB 780,820,850,890,930,970,1000,10
       30,1070,1100,1130
JA 430 READ X1,Y1,X2,Y2:LINE(X1,Y1)-(X2,Y2)
JJ 440 READ C
GB 450 FOR J=1 TO C
ND 460 READ X,Y:LINE -(X,Y)
OC 470 NEXT J
OP 480 BEEP
CJ 490 LOCATE 21,1:INPUT "STATE";SS$
CL 500 IF SS$=S$(R) THEN 590
FD 510 SOUND 165,2:SOUND 131,2
KL 520 F=F+1:IF F=2 THEN 550
NE 530 LINE (0,152)-(299,199),0,BF
```

```
JJ 54Ø GOTO 49Ø
DP 55Ø PRINT "The state is ";S$(R)
NL 56Ø PRINT:PRINT "<Press any key to continue>";
LL 57Ø A$=INPUT$(1)
DA 58Ø GOTO 22Ø
GM 59Ø SOUND 262,2:SOUND 33Ø,2
JG 6ØØ SOUND 392,2:SOUND 523,4:F2=Ø
DO 61Ø LOCATE 23,1:INPUT "CAPITAL";CC$
GH 62Ø IF CC$=C$(R) THEN 69Ø
GI 63Ø SOUND 165,2:SOUND 131,2
DB 64Ø F2=F2+1:IF F2=2 THEN 67Ø
EB 65Ø LINE (Ø,168)-(299,199),Ø,BF
FK 66Ø GOTO 61Ø
PN 67Ø PRINT "The capital is ";C$(R)
IL 68Ø GOTO 56Ø
GN 69Ø SOUND 262,2:SOUND 33Ø,2
BC 7ØØ SOUND 392,2:SOUND 523,4
LJ 71Ø S$(R)=""
AD 72Ø NEXT N
NJ 73Ø CLS:SCREEN Ø
GD 74Ø FOR C=1 TO 4Ø
LE 75Ø SOUND 5ØØ*RND+3ØØ,1
KN 76Ø NEXT C
ND 77Ø GOTO 116Ø
BP 78Ø RESTORE 79Ø:REM TEXAS
KG 79Ø DATA 13Ø,118,148,118,9,148,95,16Ø,95,16Ø,1
       Ø6,17Ø,1Ø8,183,1Ø8
FG 8ØØ DATA 187,11Ø,188,118,19Ø,122,189,13Ø
ME 81Ø RETURN
PC 82Ø RESTORE 83Ø:REM OKLAHOMA
EN 83Ø DATA 147,92,184,92,5,186,1Ø8,16Ø,1Ø6,16Ø,9
       6,147,96,147,92
NK 84Ø RETURN
DB 85Ø RESTORE 86Ø:REM ARKANSAS
BM 86Ø DATA 184,94,2Ø3,94,9,2Ø3,96,2Ø6,96,2Ø5,1ØØ
       ,2ØØ,1Ø7,2ØØ,112
NJ 87Ø DATA 187,112,187,11Ø,184,11Ø,184,94
NC 88Ø RETURN
IH 89Ø RESTORE 9ØØ:REM LOUISIANA
NJ 9ØØ DATA 2Ø9,125,2Ø8,122,7,199,122,2Ø2,116,2ØØ
       ,111,188,111
OH 91Ø DATA 188,117,19Ø,123,189,129
MH 92Ø RETURN
OI 93Ø RESTORE 94Ø:REM ALABAMA
KD 94Ø DATA 214,124,213,1ØØ,7,2Ø3,1ØØ,2ØØ,1Ø7,2ØØ
       ,111,2Ø2,115
JG 95Ø DATA 199,122,2Ø8,122,2Ø9,126
NP 96Ø RETURN
KL 97Ø RESTORE 98Ø:REM TENNESSEE
```

```
BK 980 DATA 212,90,240,90,5,228,98,202,98,206,92,
       212,92,212,90
NF 990 RETURN
NC 1000 RESTORE 1010:REM MISSISSIPPI
BG 1010 DATA 213,124,213,99,5,224,99,228,112,229,
        119,218,119,219,123
IB 1020 RETURN
KE 1030 RESTORE 1040:REM GEORGIA
OC 1040 DATA 245,110,234,97,8,224,97,228,111,227,
        115,230,119
NC 1050 DATA 242,119,243,121,243,117,246,117
JN 1060 RETURN
DL 1070 RESTORE 1080:REM FLORIDA
IH 1080 DATA 219,123,217,119,6,228,119,228,120,24
        2,120,243,121,243,117,246,117
JG 1090 RETURN
CN 1100 RESTORE 1110:REM NORTH CAROLINA
IL 1110 DATA 263,83,239,87,4,230,96,244,94,251,95
        ,256,98
ID 1120 RETURN
BE 1130 RESTORE 1140:REM SOUTH CAROLINA
EC 1140 DATA 245,110,234,97,4,244,94,246,96,251,9
        5,256,99
JM 1150 RETURN
HF 1160 END
```

Customizing the Function Keys

Melody and Michael A. Covington

Using this short and simple technique can save you lots of repetitive typing when programming in BASIC. It works equally well on the IBM PC or PCjr.

In BASIC on the IBM PC or PCjr, you can use the special function keys to type commonly used commands with a single keystroke. For example, pressing F2 is equivalent to typing RUN and hitting Enter. The definitions of the keys are displayed in a list at the bottom of the screen; the BASIC command KEY OFF erases the list but does not affect the functioning of the keys themselves. In 40-column screen modes (such as the default mode on the PCjr) only half of these definitions are visible at a time.

You can redefine a function key with a command such as

```
KEY 1, "FILES"
```

This makes key F1 equivalent to typing FILES (without Enter). To include Enter, you could do something like this:

```
KEY 1, "FILES"+CHR$(13)
```

where 13 is the ASCII code for a carriage return.

Each key can stand for up to 15 characters and can represent anything you can type—a command, a BASIC keyword, a recurrent phrase, or whatever you want. Any of the IBM's 256 characters can be included.

You can even create your own personalized set of key definitions and arrange things so that these definitions, rather than the ones supplied by IBM, are in effect every time you enter BASIC. That way, you won't have to spend any time redefining keys at the beginning of each programming session.

The trick is to begin the session by running a brief program which changes some key definitions and then obliterates itself. The accompanying program listing shows an example, called MYKEYS.BAS. After redefining four keys, the program

prints an identifying message and executes a NEW command, erasing itself from memory. This leaves you with a clean slate on which you can type your own programs.

If you save MYKEYS.BAS on your DOS disk, you can enter your customized BASIC by typing

A> BASIC MYKEYS (or BASICA MYKEYS)

instead of just BASIC or BASICA. The customized key definitions appear in the list at the bottom of the screen; the default definitions apply to any key you did not redefine.

Customizing the Function Keys

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
AG 10 REM MYKEYS.BAS
OC 20 REM Redefines function keys:
JI 30 REM F5=FILES "A:*.*" (with RETURN)
DF 40 REM F6=FILES"B:*.*" (with RETURN)
LC 50 REM F7=PRINT "  (for use in programs)
BH 60 REM F8=DATA     (for use in programs)
BF 70 REM Other functions unchanged
CK 80 QUOTE$=CHR$(34)
BA 90 CR$=CHR$(13)
OB 100 KEY 5, "files "+QUOTE$+"A:*.*"+QUOTE$+CR$
BJ 110 KEY 6, "files "+QUOTE$+"B:*.*"+QUOTE$+CR$
IC 120 KEY 7, "PRINT "+QUOTE$
BD 130 KEY 8, "DATA"
FB 140 PRINT "IBM PC/PCjr BASIC with customized k
       eys"
AC 150 KEY ON
HD 160 NEW:REM This program erases itself!
```

BASIC's Undocumented SHELL Command

Michael A. Covington

With DOS 3.0, IBM has announced a number of new features for Disk BASIC. At least one of them is actually present in DOS 2.0 and 2.1 as well, though the manuals do not mention it. That feature is a command called SHELL which allows you to execute DOS commands from within BASIC. (The technique does not work with PCjr Cartridge BASIC.)

The SHELL command in IBM BASIC takes one parameter, a character string containing the DOS command to be executed. SHELL works by loading, from drive A, a second copy of COMMAND.COM (the DOS command processor) and invoking it as a subprocess. (Note that this implies that COMMAND.COM must be present on the disk in drive A when the SHELL command is executed.) The top level COMMAND.COM and the BASIC interpreter are in suspended animation until the subprocess finishes; then control returns to BASIC.

SHELL handles the cursor somewhat awkwardly. When the SHELL command is executed, the screen is cleared from the current cursor position to the bottom; DOS writes its output there, scrolling as needed (the twenty-fifth line scrolls along with the others). But when control returns to BASIC, the cursor suddenly appears one line below where it was when the subprocess started, ignoring all screen activity that took place under the subprocess.

The best way to prevent chaos on the screen is to execute a CLS (clear-screen) immediately after each SHELL, or as soon afterward as you've finished looking at the output.

Not a Child

The one command that SHELL cannot issue, either directly or indirectly, is BASIC (or BASICA). If you try to do this, you get the message "You cannot run Basic as a Child of Basic"—naturally enough, you can't run BASIC in the subprocess because most of BASIC is in ROM and there's only one copy of

it in the machine. If you issue a SHELL, and COMMAND.COM is not on drive A, you get a "File not found" error within BASIC.

The most useful SHELL commands are probably

SHELL "A:"
SHELL "B:"

and the like, to change logged disks. These are foolproof commands; they produce no messages to clutter up the screen, and they can't terminate abnormally.

You can also use SHELL without parameters, in immediate mode, to enter the DOS command mode. The advantage of this over SYSTEM is that when you've finished issuing DOS commands, you can type EXIT and return to BASIC with your program undisturbed.

Most kinds of errors in the subprocess will return you to BASIC with no problems, but a few, such as typing A in response to "Abort, Retry, Ignore," will leave you in the DOS command level of the subprocess, in which case you must type EXIT to get back to BASIC.

One at a Time

Don't issue several SHELL commands in succession if you can avoid it; each of them loads COMMAND.COM all over again. Instead, if you have a series of commands to issue, write them onto a .BAT file from within BASIC, and give one command to run the whole file.

The accompanying program demonstrates one way to use SHELL to create a menu-driven user interface for DOS. Naturally, a practical program would include many more options and more error checking.

Purpose:	Executes a DOS command from within BASIC. This is done by loading a second copy of COMMAND.COM and invoking it as a subprocess.				
Versions:	Cassette	Disk	Advanced	Compiler	
	no	yes	yes	?	
Format:	SHELL or SHELL X$				

Remarks: X$ is a character string constant, variable, or expression containing any valid DOS command.

In order for SHELL to work, COMMAND.COM must be present on disk A. If it is not, the message "File not found" is displayed.

X$ can be an internal DOS command or invoke a .COM, .EXE, or .BAT file. However, the BASIC interpreter cannot be invoked using SHELL; if this is attempted,the message "You cannot run Basic as a Child of Basic" is displayed.

The amount of memory available in the subprocess is markedly less than is available in DOS by itself.

If X$ is omitted, the user is placed at the DOS command level of the subprocess. To return to the calling BASIC program, type the command EXIT.

Certain fatal errors in the subprocess may also leave the user at the DOS command level of the subprocess; again, typing EXIT returns control to BASIC. However, most errors in the subprocess return control to the calling BASIC program automatically.

Examples:

SHELL	(to go temporarily into command mode)
SHELL "B:"	(to change logged disk)
SHELL "DIR A:	: SORT : MORE"
SHELL "MYFIL"	(to invoke MYFIL.COM, MYFIL.BAT, or MYFIL.EXE, as the case may be)

CHAPTER 6

Demo of SHELL Command

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
GI 10 ' COMMAND.COM must be on drive A
IA 20 ' MORE.COM and CHKDSK.COM must be
ME 30 '    on the current default disk
KN 40 CLS: KEY OFF
BD 50 PRINT "Welcome to menu-driven DOS."
FI 60 PRINT
LO 70 PRINT "Available functions are:"
NG 80 PRINT "  1  Directory of disk A"
AG 90 PRINT "  2  Directory of disk B"
QJ 100 PRINT "  3  Disk and memory information"
BD 110 PRINT "  4  Copy a file"
AB 120 PRINT "  5  View a file"
JK 130 PRINT "  6  End this program"
JI 140 PRINT
KA 150 INPUT "Choose one...";N
BL 160 IF N=6 THEN CLS: END
OF 170 IF (N<1) OR (N>5) THEN BEEP: GOTO 150
BN 180 CLS
KA 190 ON N GOTO 210,240,270,320,370
MD 200 ' directory of A
AH 210 SHELL "dir a:"
CP 220 GOTO 400
ON 230 ' directory of B
CN 240 SHELL "dir b:"
CF 250 GOTO 400
OG 260 ' disk & memory info.
CG 270 INPUT "Drive to check ";A$
BI 280 IF A$="a"OR A$="A" THEN SHELL "chkdsk a:"
FG 290 IF A$="b"OR A$="B" THEN SHELL "chkdsk b:"
CM 300 GOTO 400
ML 310 ' copy a file
MC 320 INPUT "File to copy from ";A$
CM 330 INPUT "File to copy onto ";B$
EO 340 SHELL "copy "+A$+" "+B$
CG 350 GOTO 400
KD 360 ' view a file
OA 370 INPUT "Name of file ";A$
CG 380 SHELL "more <"+A$
LJ 390 ' finish up
DN 400 LOCATE 25,1
MD 410 WHILE INKEY$<>"": WEND
OB 420 PRINT "(Press any key to continue...)";
LA 430 WHILE INKEY$="": WEND
OF 440 GOTO 40
```

Unlocking BASIC Programs

Peter F. Nicholson

This short utility unlocks BASIC programs which have been saved in protected format with the P option. It works on any IBM PC or PCjr.

IBM BASIC lets you save a program in three formats: in tokenized (compressed binary) form, as an ASCII file, or as a protected (encoded binary) program. The commands for these options are

SAVE ***"filename"*** (tokenized)
SAVE ***"filename"*,A** (ASCII)
SAVE ***"filename"*,P** (protected)

In each case, DOS automatically appends the extender .BAS and does not indicate the format on disk directories. You can load a program saved in any format with LOAD "*filename*.BAS", omitting the .BAS extender if you wish.

Although a protected program loads and runs normally, it cannot be listed or edited, and neither BASIC nor DOS provides a way to "unlock" it. So when you save a program in protected format, you should also save an unprotected copy in case you decide to make some changes later. If you find yourself without a backup, however, the following utility can remove the protection.

Type in and save UNPROT.UTL. Note that *you must save it with the filename UNPROT.UTL before you attempt to run it.* When you run it, you'll be prompted to enter the active drive (enter A if you have one drive) and the name of the protected program. The drive runs briefly as the protection is removed, and then your program is listed on the screen, ready for you to edit or resave. The copy of the program on your disk is still saved in protected form; to save a copy in unprotected form you must resave the program.

Invisible Fingers

To mimic the effect of entering direct keyboard commands, UNPROT.UTL assigns strings to the ten special function keys.

It then manipulates the keyboard buffer to enter each string automatically, as if the function keys were being pressed in sequence by invisible fingers. If you use DOS 2.1, the subroutine at line 2000 automatically enters the function keystrokes for you. If you have another version of DOS, you'll have to delete the GOSUB 2000 statement from line 290 and press F1 through F10 in sequence yourself, after entering the filename.

Mimicking keystrokes is an efficient technique, but it makes a program somewhat difficult to follow. If you're interested in how this utility works, here's a brief explanation of how protected programs can be unlocked.

The Key Addresses

The first thing you need to learn is where the program starts and ends in memory. As explained in Appendix I of the *IBM BASIC Manual*, these addresses can be found with the following PEEKs:

PEEK(&H30)+256*PEEK(&H31)	Program starting address
PEEK(&H358)+256*PEEK(&H359)−1	Program ending address

The starting address is the same in every case; you can find it simply by entering NEW followed by the first PEEK statement above. Finding the ending address is more difficult, as you'll find if you load a protected program and enter the second statement. All you'll get for your trouble is an illegal function call error.

However, there's another way to get the same information. Scalar variables are stored immediately after the end of a BASIC program, and the VARPTR function can find the address of any variable. All you need to do is define an arbitrary scalar variable, CHAIN the protected program into memory, and use VARPTR to find the address of the dummy variable.

Breaking the Chains

Unlike the LOAD command, which clears variables, CHAIN brings a program into memory and begins execution at a specified line number without destroying preexisting variables. This is the method used in UNPROT.UTL. We don't want to *run* the chained program after it's in memory, so the CHAIN command uses a nonexistent line number (65529). This simply halts execution with an illegal function call error.

Subtracting a few bytes to account for the variable descriptor gives us the exact address where the program ends. To determine its length, we subtract the starting address from the ending address.

Now that we know the program's starting address and length, we BSAVE it back to disk as a binary file. After performing a second NEW, it's necessary to set the pointers for the start of scalar variables, arrays, and strings at the spot where the program ends. Finally, the program is BLOADed back into memory at the correct starting address, and the unlocking process is complete.

UNPROT.UTL

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
ID 80 REM 'UNLOCK' PROGRAMS SAVED IN PROTECTED FO
      RMAT.
MA 90 REM LOADS A PREVIOUSLY PROTECTED PROGRAM IN
      TO MEMORY
MB 95 REM WITHOUT PROTECTION, SO PROGRAM CAN BE L
      ISTED AND SAVED.
KD 96 REM IBM BASIC VERSIONS 1.1 AND 2.0
JL 100 DEF SEG:CLEAR:KEY OFF:CLS:ON ERROR GOTO 30
       0
HJ 110 B%=0:A=0
NM 120 GOSUB 1000
KD 130 A=PEEK(&H30)+256*PEEK(&H31)
IL 140 LINE INPUT "PROTECTED FILE DRIVE ";G$:IF L
       EN(G$)>0 THEN IF INSTR(G$,":")=0 THEN G$=G
       $+":"
HO 150 LINE INPUT "PROTECTED FILE NAME ";F$:IF IN
       STR(F$,".")=0 THEN F$=F$+".BAS"
HP 160 G$=G$+F$
DI 170 F$="PROT.SCR"
HG 180 H$="PROT.DAT":I$="UNPROT.UTL"
HK 190 KEY 1,"B=VARPTR(B)"+CHR$(13)
IF 200 KEY 2,"BSAVE F$,A,B-A"
GL 210 KEY 3,"-4:BSAVE H$,B,"
ML 220 KEY 4,"4:CHAIN I$,500"+CHR$(13)
ON 230 KEY 5,"BLOAD"+CHR$(34)
ME 240 KEY 6,H$+CHR$(34)+",856"+CHR$(13)
EI 250 KEY 7,"BLOAD"+CHR$(34)+F$
IH 260 KEY 8,CHR$(34)+","+STR$(A)+CHR$(13)+"LIST"
       +CHR$(13)
AC 270 KEY 9,"FOR I=1TO 10:KE"
HN 280 KEY 10,"Y I,"+CHR$(34)+CHR$(34)+":NEXT"+CH
       R$(13)
```

```
LJ 29Ø GOSUB 2ØØØ:COLOR Ø,Ø:CHAIN G$,65529!,ALL
EE 3ØØ FOR I=1 TO 1Ø:KEY I,"":NEXT I
DH 31Ø COLOR 7,Ø:IF ERL=29Ø AND ERR=53 THEN CLS:B
       EEP:PRINT G$+" DOES NOT EXIST":RESUME 14Ø
NE 32Ø ON ERROR GOTO Ø:END
GB 5ØØ B=Ø:DIM B1%(2):COLOR 7,Ø
DD 51Ø BLOAD "PROT.DAT",VARPTR(B)
DB 52Ø FOR I=Ø TO 2:IF B<2^15 THEN B1%(I)=B ELSE
       B1%(I)=B-2^16
MC 53Ø NEXT I:BSAVE "PROT.DAT",VARPTR(B1%(Ø)),6
DC 54Ø NEW
HH 1ØØØ PRINT "UNPROTECTING BASIC PROGRAMS"
OJ 1Ø1Ø LOCATE 4,1Ø:PRINT "1. YOU WILL BE PROMPTE
        D FOR THE FILE DRIVE AND NAME"
PC 1Ø4Ø LOCATE 7,1Ø:PRINT "2. THE FINAL STEP IS T
        HE LISTING OF YOUR PROGRAM"
GF 1Ø5Ø LOCATE 16,1:PRINT "NOTE: FUNCTION KEYS AR
        E CLEARED BY THE PROGRAM AND TWO SCRATCH"
KM 1Ø6Ø LOCATE 18,1:PRINT "FILES, PROT.SCR AND PR
        OT.DAT ARE LEFT ON YOUR DEFAULT DRIVE"
DH 1Ø7Ø LOCATE 25,1:PRINT "PRESS ANY KEY TO START
        ";
CB 1Ø8Ø KB$=INKEY$:IF KB$="" GOTO 1Ø9Ø: 'CLEAR KE
        YBOARD
EN 1Ø9Ø KB$=INKEY$:IF KB$="" GOTO 1Ø9Ø
EF 11ØØ CLS:RETURN
GF 2ØØØ REM ** SET KEYBOARD BUFFER TO ENTER F1 TH
        ROUGH F1Ø AUTOMATICALLY***
DM 2Ø1Ø REM **                    IBM PC DOS VERSION 2.1
                                      ***
JG 2Ø2Ø DEF SEG=&H4Ø:FOR I=1 TO 1Ø:POKE 2*I+28,Ø:
        POKE 2*I+29,58+I:NEXT I
CO 2Ø3Ø POKE 2*I+28,13:POKE 2*I+29,28:POKE 26,3Ø:
        POKE 28,5Ø:DEF SEG:RETURN
```

CHAPTER 7

Machine Language

PEEKs and POKEs

Michael A. Covington

Do you yearn to roam beyond the bounds of BASIC? With a few PEEKs and POKEs, you can perform useful tricks that aren't normally possible in your computer's native language. Most of them work on any PC or PCjr.

IBM Microsoft BASIC makes a remarkably wide range of functions available to the programmer. There are statements to trap errors (ON ERROR GOTO), statements to control the screen (CLS, LOCATE), a statement to deallocate arrays which are taking up too much space (ERASE), and even a statement to scan input ports for activity (WAIT). As a result, IBM programmers don't need to use nearly as many PEEKs and POKEs as their counterparts on other microcomputers.

But there are still a few functions which, although not built into BASIC, can be squeezed out of the computer with PEEKs and POKEs. This article will present a sampling of them. Remember, however, that programs which use machine-specific PEEKs and POKEs may not work on other PC-compatible computers or even on various models of the same machine.

Segments and Offsets

Unlike 8-bit computers, the PC and PCjr are capable of directly addressing a full megabyte of memory (1024K, or 1,048,576 bytes). But an ordinary 16-bit address can have only 65,536 different values, sufficient for 64K of memory at most. IBM's solution is to specify each memory address in two parts, a *segment* and an *offset,* related as follows:

Absolute address = (16 * *segment*) + *offset*

With this system, a given address can be referred to in more than one way. For instance, segment 0, offset 16 works out to the same address as segment 1, offset 0 (absolute address 16).

In BASIC, the PEEK function and the POKE statement each specify only the offset. The segment is specified by a

DEF SEG (*default segment*) statement. For example, here's how you might examine absolute address 16:

10 DEF SEG = 0
20 PRINT PEEK(16)

Each DEF SEG remains in effect until another one is executed. A DEF SEG without an argument (that is, with no = sign and segment number) returns the default segment to the one the BASIC interpreter is using.

It's often convenient to specify PEEK and POKE addresses as hexadecimal numbers (base 16). In IBM BASIC, a hexadecimal number is preceded with &H. For example, &HFF equals 255 in decimal. (The hexadecimal digits are 0–9, A, B, C, D, E, and F.) Internally, such numbers are treated as integers. You can do Boolean operations on the individual bits with the BASIC operators AND, OR, and NOT.

Snooping Around

Let's start by collecting some trivial information about the computer. One of the ROM (Read Only Memory) chips inside the PC and PCjr has two copyright dates burned into it. Here's how to read them.

```
EG 1Ø  DEF SEG = &HFFFF
MD 2Ø  FOR X=1 TO 5
GC 3Ø  PRINT CHR$(PEEK(X));
QJ 4Ø NEXT X
EH 5Ø PRINT
JB 6Ø DEF SEG=&HFØØØ
GA 7Ø FOR X=&HEØØØ TO &HEØ15
GH 8Ø  PRINT CHR$(PEEK(X));
QO 9Ø NEXT X
IA 1ØØ PRINT
```

If you enter this program on a PC and run it, you should get something like the following:

10/27/82
1501476 COPR. IBM 1982

On a PCjr, we got this result:

C = 0
1504037 COPR. IBM 1981

The exact dates, however, may be different; two commonly encountered ones are 04/21/81 and 10/19/81. (You might say that ROM wasn't built in a day.)

C H A P T E R 7

You can discover additional things about the computer with this short program:

```
GA 1Ø DEF SEG=&H4Ø
BP 2Ø MEMORY=PEEK(&H13)+256*PEEK(&H14)
DG 3Ø CRTCOLS=PEEK(&H4A)
BJ 4Ø CRTMODE=PEEK(&H49)
BN 5Ø PRINT "memory available";MEMORY;"K"
OJ 6Ø PRINT "crtmode is";CRTMODE
```

Here, MEMORY is the amount of installed memory, in kilobytes; CRTCOLS is the width of the screen (40 or 80); and CRTMODE is a code indicating the type of display (0 to 6 for the various graphics modes of the color/graphics adapter, and 7 for the IBM Monochrome Display). It can be especially useful for some programs to use CRTCOLS and CRTMODE to determine what kind of screen they are running on. It seems to work equally well on a PC or PCjr.

Changing Displays

The following program fragments, from the IBM BASIC manual, let you switch between the monochrome and color display adapters, if you have both:

```
LO 1Ø ' Switch to monochrome
JJ 2Ø DEF SEG=Ø
AD 3Ø POKE &H41Ø,(PEEK(&H41Ø) OR &H3Ø)
DE 4Ø SCREEN Ø:WIDTH 4Ø:WIDTH 8Ø
FF 5Ø LOCATE ,,1,12,13
```

```
DI 1Ø  ' Switch to color/graphics
JJ 2Ø DEF SEG=Ø
HE 3Ø POKE &H41Ø,(PEEK(&H41Ø) AND &HCF) OR &H1Ø
JO 4Ø SCREEN 1,Ø,Ø,Ø:SCREEN Ø:WIDTH 4Ø
MA 5Ø LOCATE ,,1,6,7
```

What you're doing here is altering the value of a special memory location (at absolute addresses &H410 and &H411) which tells the PC what kind of hardware is installed. The subsequent SCREEN, WIDTH, and LOCATE statements are necessary to properly initialize the newly selected display before you use it. If you run either of these routines on a PC without the adapter you're trying to switch to, the system will hang up, since the hardware description stored in &H410 will not match the actual hardware. If you run the monochrome

routine on a PCjr—which has the equivalent of a built-in color/graphics adapter—the screen switches into the 80-column mode (on Enhanced Models) and the cursor inexplicably disappears.

You can use the hardware descriptions in &H410 and &H411 for other purposes as well. Here is a routine that tells you how many printers, joystick ports (0 or 1), RS-232 ports, and disk drives the PC is equipped with:

```
JI 10 DEF SEG=0
MG 20 PRINTERS=(PEEK(&H411) AND &HC0)/64
MF 30 JOYPORTS=(PEEK(&H411) AND &H10)/16
GO 40 RS232PRT=(PEEK(&H411) AND &HE)/2
KA 50 HAVEDISK=(PEEK(&H410) AND 1)
JH 60 DISKDRIV=HAVEDISK*(1+((PEEK(&H410) AND &HC0
      )/64))
```

HAVEDISK equals 1 if the PC has at least one disk drive, and 0, otherwise. On the PCjr, JOYPORTS equals 1 even though two joystick ports are standard, and DISKDRIV equals 0 even when HAVEDISK is 1.

These hardware descriptions are initialized when the computer is first turned on. On the PC, they reflect the presence of the appropriate adapter cards and/or switch settings inside the system unit, but they do not indicate whether outside equipment (for example, a printer) is powered on or even connected.

Manipulating the Keyboard

The PC and PCjr have a built-in 15-character keyboard buffer which allows you to get ahead of the program that is reading the characters you type. (If you get ahead by more than 15 characters, the beeper sounds.) This makes it practical to work very fast by anticipating what the computer is going to ask you next, even though the prompt may not have appeared yet.

But when the program takes an unexpected turn—if it detects an error, for instance—it may be desirable to discard any unprocessed characters in the buffer. They were typed in anticipation of prompts that aren't going to be displayed. Use these statements:

```
JI 10 DEF SEG=0
QL 20 POKE 1050,PEEK(1052)
```

The computer keeps track of the memory addresses where the active part of the keyboard buffer begins and ends; this operation simply sets the end equal to the beginning.

There is, of course, a less machine-dependent way to do the same thing, though it takes a fraction of a second longer:

```
OJ 10 IF INKEY$<>"" THEN 10
```

This simply reads characters from the keyboard, one by one, until there are no more left.

You can toggle the Num Lock and Caps Lock keys on both the PC and PCjr by means of PEEKs and POKEs. For example:

```
LC 10 ' Enter Num Lock mode
GB 20 DEF SEG=&H40
LH 30 POKE &H17,(PEEK(&H17) OR &H20)
```

```
PF 10 ' Exit Num Lock mode
GB 20 DEF SEG=&H40
JC 30 POKE &H17,(PEEK(&H17) AND (&HFF-&H20))
```

You can enter and exit Caps Lock mode with similar statements, using &H40 in place of &H20 each time it occurs in the listing above. Not only does this simplify typing, it also rescues the user from confusing situations which might result if the state of Caps Lock or Num Lock is unknown when the program starts.

Reading the Timer

In BASIC you can use the pseudovariable TIME$ to get the time of day in hours, minutes, and seconds (for example, "08:23:14"). But you can use PEEKs to access the timer in "raw," undecoded form as a number which repeatedly counts from 0 to 65535 and is incremented 18 times per second. This raw timer value is useful as a random number seed. For example:

```
GA 10 DEF SEG=&H40
NO 20 CLOCK=PEEK(&H6C)+256*PEEK(&H6D)
DC 30 RANDOMIZE CLOCK-32768!
```

This is equivalent to what most other computers do when executing the RANDOMIZE statement by itself; they do not ask you for a random number seed the way the IBM does.

Going Deeper

I worked out most of these PEEKs and POKEs from the ROM BIOS listings in the IBM PC *Technical Reference Manual*, which is an essential guide for advanced programmers who want to perform machine-dependent functions on the PC. A similar manual is available for the PCjr. But the manuals assume a rather advanced knowledge of microcomputer design. If you're relatively new to the field, you may want to read Peter Norton's book *Inside the IBM PC* (Brady, 1983). And from COMPUTE! Books, *Mapping the IBM PC and PCjr*, by Russ Davies, is an excellent guide to IBM memory.

The 8088 Microprocessor: Brains of PC and PCjr

Ottis R. Cowper

IBM advertises the PC and PCjr as 16-bit computers built around the powerful 8088 microprocessor. But Intel, the manufacturer of the chip, calls the 8088 an 8-bit processor. How do you build a 16-bit microcomputer using an 8-bit microprocessor?

The 8088 has something of an identity crisis. Circuit designers classify microprocessors on the basis of the number of *bits* of data that enter or exit the chip at a time. On the 8088, the data bus (the set of wires for data signals) consists of 8 bits (eight wires), so Intel calls it an 8-bit processor. The 8088 also generates most of the same control signals as Intel's 8085, a widely used 8-bit processor. On the other hand, data *within* the 8088 is manipulated in 16-bit chunks. The processor's internal memory locations (registers) are 16 bits wide. The arithmetic logic unit (ALU), the circuitry which performs arithmetic functions, works with 16 bits at a time. (The fancy term for this is *16-bit internal architecture.*) The 8088 also uses the same machine language instructions as Intel's 8086, a true 16-bit microprocessor. Thus, IBM is not being totally untruthful when it calls PC and PCjr 16-bit machines.

In the Middle

The 8088 is in the middle ground between the older 8-bit microprocessors and the newer 16-bit chips. It is sufficiently similar to 8-bit processors that the hardware designers for the PC and PCjr were able to easily adapt existing circuitry into their designs. However, the 8088 can very nearly match the performance characteristics of the true 16-bit processors. Its expanded instruction set supports advanced operations not commonly available on 8-bit units, such as string processing and integer multiplication and division.

Most 8-bit microprocessors have 16 memory address lines. This 16-bit *address bus* allows the processor to communicate directly with 2^{16}, or 65536 (64K), bytes of memory. The 8088

has a 20-bit address bus. With 20 address signal lines, the processor can communicate directly with 2^{20}, or 1,048,576, bytes (called a megabyte). This memory is divided into 16 segments with 64K per segment. Thus, it's obvious that the 8088 can easily control much more memory than an 8-bit processor.

With many microprocessors—for example, the 6502 used in the Apple, Commodore, and Atari—any input/output (I/O) chips such as video controllers and sound generators connected to the system occupy memory space. That is, to communicate with these *memory-mapped* I/O devices, the microprocessor must address the device registers as if they were regular data storage locations. The 8088 addresses I/O devices independently of other memory so that adding such devices does not take away from the amount of data storage space available. In fact, the 8088 can directly address quite a lot of I/O devices, 65536 to be exact. Compare this with the 255 devices which can be addressed by its true 8-bit cousins, the 8080 and 8085 microprocessors.

Instruction Pipeline

Another advanced feature of the 8088 is instruction pipelining. In conventional designs, the microprocessor must *fetch* a machine language instruction from memory, *execute* that instruction, fetch another instruction, execute that instruction, and so on. If the 8088 had been designed this way, more fetching would be required for each execution, since it is necessary to transfer data into the 16-bit internal bus of the processor through its 8-bit data bus to the outside world. This would make for very inefficient operation.

To get around this bottleneck, the 8088 is divided internally into two parts, the bus interface unit (BIU), which transfers information to and from memory, and the execution unit (EU), which performs the required operations on the information. While the EU is doing its work on the data, the BIU is busy getting more data from memory and stuffing it into an instruction queue. When the EU finishes executing one instruction, it usually has to look no further than the queue to get the next instruction. Very little time is wasted waiting for data to be pulled from memory into the processor for execution. This makes it possible for the 8088 with its 8-bit data bus to operate almost as fast as if it had a 16-bit data bus to match its 16-bit internal architecture. It also means that the processor

can operate at high speeds without requiring more expensive, fast-responding memory chips.

The 8088 has a family of advanced support chips to further enhance its performance. For example, the processor is most commonly used in what is called *minimum mode* configuration. However, it can be set up for *maximum mode*, in which the processor can share control of the address and data buses with other devices. Configuring the processor for maximum mode allows an 8087 math coprocessor to be added. This chip adds more than 60 commands to the 8088 instruction set and will perform 80-bit floating-point arithmetic operations and transcendental functions such as tangent and logarithm directly. (The 8088 itself can do only integer arithmetic internally.) Intel claims that the 8087 will allow the system to accomplish arithmetic operations 100 times faster than the 8088 could perform them in software. The PC allows the 8088 to be switched to maximum mode and has a socket on its main circuit board in which an 8087 can be installed. It is not possible to add an 8087 to the PCjr because the 8088 on Junior's main circuit board is wired for minimum mode configuration only.

The 8088 is a microprocessor of compromise. Intel developed it to bridge the gap between the 8-bit and 16-bit worlds, and endowed it with many of the features of both. Whether your PC/PCjr is truly an 8-bit or a 16-bit computer remains a subject you can debate with your neighborhood PC hacker.

Experiments in Machine Language

Michael A. Covington

If your programming experience is limited to BASIC, this article presents a simple introduction to machine language programming. With an Enhanced Model PCjr or a PC with a disk drive, you can write short machine language programs using only the utilities included on your DOS and Supplemental Programs *disks.*

When you write and run a program in BASIC, you're not really giving instructions directly to the 8088 microprocessor inside your computer. The microprocessor doesn't understand BASIC any more than it understands English. The only language it really understands is a series of numbers known as *machine language.*

The BASIC program you write actually consists of statements which are translated one at a time into machine language by an interpreter program. The interpreter program itself is written in machine language—and it's called BASIC. BASIC is like a foreign language interpreter which allows you to communicate with the computer in its own native tongue.

Thus, your BASIC programs are always a step away from actually giving instructions directly to the computer. This distance isn't too apparent unless you're trying to write a program which must run very fast. Then it becomes obvious. The delay caused by the translation process from BASIC into machine language makes the computer appear slow.

If you could write the program in machine language to begin with, instead of in BASIC, no translation would be necessary. The computer could run at full speed.

That's what machine language programming is all about, and that's why machine language is inherently much faster than BASIC. At times, machine language can also be more flexible. There are things you can do in machine language that simply can't be done in BASIC—or at least, not as directly. However, BASIC is usually easier to work with. Machine language programming can be very tedious.

Still, there are some applications that cry out for machine language. If you've never tried it, you can experiment with some short routines to acquaint yourself with this new world. Best of all, you won't have to buy anything extra—everything you need to get started came with your Disk Operating System (DOS).

Interpreters, Compilers, and Assemblers

As we mentioned, BASIC is an interpreter program. It takes your BASIC program statements and translates them into machine language instructions one at a time.

There's another kind of BASIC, too. A BASIC *compiler* also translates BASIC statements into machine language instructions, but in a different way. Rather than translating them one at a time as the BASIC program runs, a compiler translates, or *compiles,* the statements before running the program. When the statements are all compiled into machine language, the resulting instructions are stored on disk as a separate program file. Then you can run this compiled version as if it were a program written in machine language to begin with.

A compiled BASIC program usually runs much faster than an interpreted BASIC program. For most purposes it's plenty fast enough. But it's still not as fast as a program written directly in machine language. That's because a compiler isn't 100 percent efficient. You can nearly always write a program directly in machine language that's more streamlined than the compiler's translated version.

Not all compilers are BASIC compilers, of course. There are Pascal compilers, C compilers, and many others. But all work on the same principle.

There's yet another type of language sometimes called *assembly language.* For all practical purposes, you can consider assembly language synonymous with machine language. Assembly language is something you create with an *assembler,* and an assembler is basically just a utility program that makes it easier to write machine language. Instead of programming directly with machine language code numbers, the assembler lets you use simple instructions such as MOV, JMPS, and OUT. Then it translates the instructions into the machine language codes.

Your DOS disk contains a mini-assembler called DEBUG. It's not nearly as sophisticated as a full-fledged commercial assembler, but it's enough to get you started.

CHAPTER 7

Bits and Bytes

Inside your computer, data is organized into *bytes*, each comprising eight *bits*. The contents of a byte can be thought of either as a character or as a number in the range 0–255. For example, the character C and the number 67 are, on the byte level, the same.

A machine language program is simply a sequence of bytes stored somewhere in memory. Each byte represents an instruction or an item of data that an instruction can refer to. To run a machine language program, the processor needs only to be told which byte is the first instruction; after that, it reads the instructions one by one and executes them.

The key to writing short machine language programs is to use the many subroutines that are already built into the computer's operating system—both in the BIOS (Basic Input/Output System) permanently recorded on a ROM chip inside the machine, and in the DOS that you load from disk.

Each subroutine is accessed by means of an *interrupt*—a special instruction that tells the processor to suspend whatever it is doing, look up the memory address associated with a particular interrupt number, perform the instructions found there, and then (if applicable) return to the original task.

The following program performs just one interrupt; it sends a form-feed character to the printer. The program consists of only ten bytes, whose numeric values are shown at the left; the rest of the listing is commentary.

186,12,0 Move the character (ASCII 12) into DL; move 0 into DH.

184,0,5 Move 0 into AL; move 5 (code for the service requested) into AH.

205,33 Request DOS services.
(This can be done over and over to send any sequence of characters to the printer.)
Exit to DOS.

205,32 Perform interrupt number 32.

The first two instructions move data into *registers*, special storage areas within the microprocessor. Interrupt service routines look at the data in the registers to find out precisely what they are being asked to do. The 8088 has a number of registers; most commonly used are the general-purpose registers AX, BX, CX, and DX, each of which holds 16 bits. Since

instructions are one byte long, you often need only half a register, so each of these registers is divided into H (High) and L (Low) halves. The half-registers are known as AH, AL, BH, BL, CH, CL, DH, and DL.

Using the Half-Registers

The program uses interrupt 33, which looks at the value of the AH register to identify the service being requested—in this case 5, a request for printing. The character to be printed is specified in the DL register. We're using machine instructions that move values into both halves of each register in a single step; we move zeros into the unused half-registers. This isn't the most efficient way to do it, but it does reduce the number of instructions we have to know about.

So the first instruction is decimal 186, which says, "Move the next two bytes into DL and DH, respectively." The next two bytes are 12 (the ASCII code for a form-feed) and 0. Then comes a 184, which moves 0 and 5 into AL and AH. The 205 instruction performs an interrupt; the byte immediately after it gives the interrupt number, 33.

When the interrupt service routine has finished its work, control returns to the next statement in the program. At this point we could move another value into DL and perform another interrupt to print another character. But instead, let's end the program by performing interrupt 32, which returns control to DOS.

If the interrupt 32 instruction were not there, the processor would continue reading whatever bytes followed in memory, interpreting them as instructions and trying to execute them. The results would be unpredictable, but the processor would probably fall into an endless loop, making it necessary to restart the computer.

Saving the Program

Now that we have a program, let's save it to disk. The simplest way is to write the program into a file whose name ends in .COM, then execute the file by typing its name as a DOS command. DOS copies the program into memory and transfers control to it. (This is how you run BASIC.COM, CHKDSK.COM, EDLIN.COM, and other machine language programs that are provided with DOS.)

The next program is a BASIC program that creates .COM files. You put the numeric values of the bytes of your machine language into its DATA statements. You can encode the form-feed program, for example. Run this BASIC program to create a file (let's call it FORMFEED.COM), get into DOS, type FORMFEED, and your printer should advance to a new sheet of paper.

```
NL 10 REM  All-purpose file creation program
BD 20 INPUT "Name of file to create";F$
JB 30 OPEN F$ FOR OUTPUT AS #1
BP 40 READ X
KJ 50 IF X=-9999 THEN END
GK 60 PRINT #1, CHR$(X);
HD 70 GOTO 40
HD 80 REM Your DATA statements go here
IH 90 REM specifying the numeric value
DC 100 REM of each byte you want to write.
JD 110 DATA 186,12,0
HF 120 DATA 184,0,5
II 130 DATA 205,33
HL 140 DATA 205,32
FN 150 DATA -9999 : REM do not remove this stmt
```

Whereas the first sample program performed a single operation—advancing the paper on a printer—a computer is best used to perform repetitive tasks.

The next program turns your PC into a typewriter by repeatedly invoking an interrupt service routine which gets a character from the keyboard (with echoing to the screen), moves the character into a different register, and then sends it to the printer.

	Ask DOS to get a character from the keyboard.
184,0,1	Move 0 into AL; move 1 (code for keyboard input) into AH.
205,33	Request DOS services.
	Print the character.
139,208	Copy AL to DL and AH to DH.
184,0,5	Move 0 into AL; move 5 (code for printer output) into AH.
205,33	Request DOS services. If you don't have a printer, replace this with 144,144 (two null operations).
	Go back to the beginning.
235,242	Go back 12 bytes (242 = 254 − 12).

Notes:
1. After each carriage return you must type a linefeed (Ctrl-J) to start a new line.
2. This program doesn't end with an INT 32 because it is an endless loop, which you can break out of with Ctrl-Break.

Machine Language Loops

The tricky thing about this program is that it involves a loop; after accepting and printing each character, it goes back to accept another one. This is equivalent to a GOTO statement in BASIC. But in machine language you must specify the exact distance in bytes that the jump must cover. In this case, the jump instruction is 235; the number following it is 254 minus the distance (12 bytes).

If you don't have a printer, you can adapt this program so that you type only on the screen. Take out the interrupt instruction that prints the character. You'll have to refigure the jump instruction, since the distance of the jump will change. As an alternative, instead of taking out the bytes you don't want, you can replace each of them with instruction 144, an instruction that performs no operation.

Not all machine language operations can be performed with numbers less than 256. To illustrate the use of larger numbers, look at the next program, which produces a beep from the speaker. No interrupt is provided for this purpose in the computer. Instead, the program sends instructions directly to the timer chip and speaker drive circuitry through the output ports:

	Inform the timer chip we are about to set frequency.
184,182,0	Move 182 into AL, 0 into AH.
230,67	Transmit contents of AL to port 67.
	Set a frequency of 1,193,000 / 1024 hertz. To do this we transmit 0 and then 4, since 1024 = 0 + 4*256.
184,0,0	Move 0 into AL, 0 into AH.
230,66	Transmit contents of AL to port 66.
184,4,0	Move 4 into AL, 0 into AH.
230,66	Transmit contents of AL to port 66.
	Turn sound on.
184,79,0	Move 79 to AL, 0 to AH.

230,97	Transmit contents of AL to port 97.
	Kill time.
244,244 **244,244** **244**	Halt five times. Each time, the processor is restarted by the timer about 1/18 second later.
	Turn sound off.
184,76,0	Move 76 to AL, 0 to AH.
230,97	Transmit contents of AL to port 67.
	Exit to DOS.
205,32	Perform interrupt number 32.

The timer chip clock frequency of 1.193 megahertz is divided by a number that we specify in order to obtain the tone frequency. In the example, the number specified is 1024, corresponding to a frequency of about 1165 hertz. But a single byte can hold only numbers between 0 and 255, so two bytes are used. To obtain the original number, the value of the high byte is multiplied by 256 and added to the value of the low byte. Since $1024 = 4 * 256 + 0$, the values needed for the high and low bytes are 4 and 0, respectively.

After setting the frequency and turning on the sound, we need to make the processor kill some time before turning the sound off again. This is done with the Halt instruction (244), which stops the processor until the PC's internal clock restarts it approximately 1/18 second later. As shown, the program produces only a brief "pip" of sound (about 1/4 second); you can add more halt instructions to lengthen it. Also, you can add instructions to load another frequency and play several notes.

Assembly Language

With programs more than about 100 bytes long, it is impractical to program in machine language directly; the computations needed for jumps and for retrieving data from memory become unmanageable. For longer programs, you will find assembly language much more efficient.

Assembly language allows you to refer to the locations of statements and data by means of symbolic names (as shown in the leftmost column of Table 1). The assembler computes the exact addresses and distances for you. Good assemblers also include such features as *macros*, prepackaged sequences of assembly language instructions that you can include in a program at any point.

C H A P T E R 7

Table 1 describes a few of the 8088 machine instructions. The table lists the name of each instruction (as abbreviated in assembly language), the numeric value in hexadecimal and decimal, and a short description. Table 2 lists a few useful interrupts.

Table 1. Selected ML Routines

Name	Hex	Decimal	Description
HLT	F4	244	Halt. On the PC, this stops the processor until the next timer interrupt about 1/18 second later.
IN xx,AL	E4	228	Input one byte into AL from an input port. Next byte gives port number.
INT	CD	205	Perform an interrupt. Next byte gives interrupt number.
JMPS	EB	235	Jumps back to an earlier instruction. The value of the following byte should be 254 − N, where N is the number of bytes you want to go back (e.g., 254 jumps back 0 bytes, to the jump instruction itself). N cannot exceed 127.
MOV AX,xx	B8	184	Move next two bytes into AL and AH respectively.
MOV CX,xx	B9	185	Move next two bytes into CL and CH respectively.
MOV DX,xx	BA	186	Move next two bytes into DL and DH respectively.
MOV BX,xx	BB	187	Move next two bytes into BL and BH respectively.
MOV DX,AX	8B	139	Moves AH into DH and AL into DL. A two-byte instruction.
	D0	208	
NOP	90	144	No Operation. Does nothing; can be inserted into a program to make its length come out right.
OUT xx,AL	E6	230	Output contents of AL to an output port. Next byte gives port number.

Note: All programs must end with an Interrupt 32 (i.e., 205 32) in order to return to DOS.

Table 2. Selected PC Interrupts

Interrupt Number Hex	Decimal	
05	05	Prints the screen (same as Shift-PrtSc).
10	16	Does various things to the screen, depending on registers. Examples: AH=6, AL=0, BH=7, BL=0, CH=0, CL=0, DH=24, DL=79 Clear entire screen. AH=9, BH=0, BL=7, CH=0, CL=1 Writes the contents of AL to the screen as one character. CL gives the number of times the character is to be repeated (usually one).
19	25	Reboots the system (like Ctrl-Alt-Del).
20	32	Terminates (returns control to DOS).
21	33	Requests any of various services from DOS, depending on contents of AH. Examples: AH=0—Terminates (like INT 32). AH=1—Keyboard input. Waits for a character to be typed and returns it in AL. Checks for Ctrl-Break. AH=2—Screen output. Displays the contents of DL (as one character). Checks for Ctrl-Break. AH=5—Printer output. The character in DL is output to LPT1.

If you want to learn more about assembly language, you may want to take a look at *COMPUTE!'s Beginner's Guide to Machine Language on the IBM PC & PCjr* by Christopher D. Metcalf and Marc B. Sugiyama. It presupposes no prior knowledge of machine language and includes sections on using interrupts and DOS functions. For a more complete guide to IBM memory, you'll want *COMPUTE!'s Mapping the IBM PC and PCjr* by Russ Davies.

You can write and assemble programs using DEBUG, but a good assembler like the *IBM Macro Assembler* makes the job a lot more convenient.

Source Code for Chess

John Krause

"Chess," in Chapter 2, is written in BASIC and machine language. Below is the commented source listing for the machine language section of Chess. The machine language takes care of the time-consuming task of thinking: It looks at the position of all the pieces and selects the best move.

```
;CHESS
;John Krause
;(c) MCMLXXXV COMPUTE! Publications, Inc.
;
;
page ,132
;
;reserve 128 words for the stack
;
stack   segment stack 'stack'
stk     dw 128 dup (?)
stack   ends
;
;
;data tables & label definitions
;
data    segment 'data'
n_moves db 21,12,248,237,235,244,8,19
q_moves db 11,247,245,9,10,1,246,255
value   db 46,9,5,3,3,1,0,1,3,3,5,9,46
piece   dw pawn,knight,bishop,rook,queen,king
level   dw ?
side    db ?
capture db 8 dup (?)
m       db 8 dup (?)
n       db 8 dup (?)
o       db 8 dup (?)
pos     db 8 dup (?)
move    db 8 dup (?)
from    db ?
to      db ?
random  db ?
score   db 8 dup (?)
board   db 120 dup (?)
test    db ?
legal   db ?
```

```
sstore dw ?
spstore dw ?
data    ends
;
;
;start of program
;
code    segment
        assume cs:code,ds:data,ss:stack
start   proc far
        push ds                     ;save data segment
        mov ax,data                 ;get location of data
        mov ds,ax                   ;change data segment
        mov ssstore,ss              ;save stack segment
        mov spstore,sp              ;save stack pointer
        mov ax,stack                ;get location of stack
        mov ss,ax                   ;change stack segment
        mov sp,size stk             ;change stack pointer
        call chess                  ;think
        mov ss,ssstore              ;restore stack segment
        mov sp,spstore              ;restore stack pointer
        pop ds                      ;restore data segment
        ret                         ;return to BASIC
start   endp
;
;
chess   proc
;
;initialization
;
        cli                         ;disable interrupts
```

```
         mov cx,8                   ;start with low scores
ca:      mov bx,cx
         mov score[bx-1],ØCØh
         loop ca
         mov random,Ø               ;low random score
         mov legal,Ø                ;clear legal flag
         mov ax,Ø                   ;ax=Ø
         mov di,ØFFFFh              ;di=-1
         jmp main                   ;start thinking
;
;move back
;
back:    mov al,pos[di]             ;get FROM position
         add al,move[di]            ;calculate TO position
         push ax                    ;save it
         mov bl,al
         mov al,board[bx]           ;get piece on TO position
         mov bl,pos[di]             ;get FROM position
         mov board[bx],al           ;move piece back to FROM position
         mov al,capture[di]         ;get captured piece
         pop bx                     ;get TO position
         mov board[bx],al           ;restore captured piece
;
;evaluate move
;
         add al,6                   ;add 6 to captured piece
         push ax                    ;save it
         mov bx,level               ;get level
         mov score+1[bx],Ø          ;score of highest ply=Ø
         pop bx                     ;get captured piece
         mov cl,value[bx]           ;get value of captured piece
```

```
        sub cl,score+1[di]      ;subtract score of next higher ply
        mov score+1[di],Øc0h    ;reset to low score
        cmp di,Ø                ;is this the first move?
        jne bla                 ;if not, just check score
        cmp cl,score            ;is this score better?
        jl e                    ;if worse, reject move
        jne d                   ;if better, skip random
;
;tie--choose at random
;
        mov al,Ø                ;get random number
        out 43h,al
        in al,4Øh
        in al,4Øh
        cmp al,random           ;is this number higher?
        jb e                    ;if lower, reject move
        mov random,al           ;highest so far
d:      cmp test,Ø              ;check human's move?
        je t                    ;if no, don't check
;
;check for illegal move by human
;
        mov al,pos              ;get computer's FROM position
        cmp al,from             ;same as human's?
        jne e                   ;if not, try another move
        add al,move             ;get computer's move
        cmp al.to               ;same as human's?
        jne e                   ;if not, try another move
        cmp cl,229              ;does move put king in check?
        jle e                   ;if so, try another move
        inc legal               ;move is legal
```

```
        ret
;
;best move so far
;
t:      mov score,cl            ;save score
        mov cl,pos              ;get FROM position
        mov from,cl             ;save it
        mov cl,move             ;get move
        mov to,cl               ;save it
e:      ret
bla:    cmp cl,score[di]        ;is score better?
        jle e                   ;if not, reject move
        mov score[di],cl        ;best score so far
;
;pruning
;
        mov bl,capture-1[di]    ;get previous captured piece
        add bl,6                ;add 6
        mov al,value[bx]        ;get its value
        sub al,score[di]        ;subtract present score
        cmp al,score-1[di]      ;is result less than previous score?
        jl bk                   ;if so, reject previous move
        cmp di,1                ;is this ply 1?
        je e                    ;if so, don't prune further
        cmp al,score-1[di]      ;is result equal to previous score?
        je bk                   ;if so, reject previous move
        ret
;
;move forward
;
forward:mov cl,pos[di]          ;get FROM position
```

```
        add cl,move[di]         ;calculate TO position
        mov bl,cl
        mov al,board[bx]        ;get piece on TO position
        cmp side,Ø              ;whose turn is it?
        jne g                   ;jump if upper player's turn
        cmp al,1                ;is piece friendly or position off board?
        jge e                   ;if so, reject move
        jl h
g:      cmp al,Ø                ;is piece friendly?
        jl e                    ;if so, reject move
        cmp al,7                ;is position off board?
        je e                    ;if so, reject move
h:      mov capture[di],al      ;save captured piece
;
;check
;
        cmp al,6                ;is captured piece lower player's king?
        je bc                   ;if so, reject previous move
        cmp al,25Ø              ;is captured piece upper player's king?
        jne bd                  ;if not, make move
bc:     mov score[di],46        ;score is high
bk:     pop dx                  ;reject previous move
        pop dx
        jmp bf
bd:     mov bl,pos[di]          ;get FROM position
        mov al,board[bx]        ;get piece to move
        mov board[bx],Ø         ;clear FROM position
        mov bl,cl               ;get TO position
        mov board[bx],al        ;put piece on TO position
;
;main loop
```

```
;
main:   cmp di,level            ;is present ply equal to level?
        jne za                  ;if not, look ahead further
        jmp back                ;if so, stop and evaluate move
za:     inc di                  ;next ply
        mov pos[di],2Ø          ;initialize FROM position
        xor side,1              ;switch sides
i:      inc pos[di]             ;next FROM position
        mov bl,pos[di]          ;get FROM position
        mov bl,board[bx]        ;get piece on FROM position
        cmp side,Ø              ;whose turn is it?
        jne ba                  ;jump if upper player's turn
        cmp bl,1                ;friendly piece on square?
        jl j                    ;if not, next position
        cmp bl,7                ;is position off board?
        je j                    ;if so, next position
        jmp l
ba:     cmp bl,Ø                ;friendly piece and on board?
        jge j                   ;if not, next position
        neg bl                  ;change piece to positive
l:      sal bl,1                ;multiply by 2
        call piece[bx-2]        ;move piece
j:      cmp pos[di],98          ;tried all positions?
        jl i                    ;if not, next position
bf:     cmp di,Ø                ;all moves tried?
        je end                  ;if so, end program
        xor side,1              ;switch sides
        dec di                  ;backtrack
        jmp back                ;and evaluate move
;
;end
```

```
;
end:    mov al,from              ;get best FROM position
        add to,al                ;calculate best TO position
        sti                      ;enable interrupts
        ret
;
;pawn
;
pawn:   cmp side,0               ;whose turn is it?
        jne aha                  ;jump if upper player's turn
        mov bl,pos[di]           ;get FROM position
        add bl,10                ;calculate position one square forward
        cmp board[bx],0          ;is square empty?
        jne ai                   ;if not, can't move forward
        mov move[di],10          ;store move
        call forward             ;move forward one square
        mov bl,pos[di]           ;get FROM position
        cmp bl,39                ;is pawn on home square?
        jge da                   ;if not, can't move two squares
        add bl,20                ;calculate position two squares forward
        cmp board[bx],0          ;is square empty?
        jne ai                   ;if not, can't move there
        mov move[di],20          ;store move
        call forward             ;move forward two squares
ai:     mov bl,pos[di]           ;get FROM position
da:     add bl,9                 ;calculate position diagonally left
        cmp board[bx],0          ;is enemy piece there?
        jge aj                   ;if not, can't move there
        mov move[di],9           ;store move
        call forward             ;move diagonally left
aj:     mov bl,pos[di]           ;get FROM position
```

```
        add bl,11               ;calculate position diagonally right
        cmp board[bx],Ø         ;is enemy piece there?
        jge ak                  ;if not, can't move there
        mov move[di],11         ;store move
        call forward            ;move diagonally right
ak:     ret
aha:    mov bl,pos[di]          ;get FROM position
        add bl,246              ;calculate position one square forward
        cmp board[bx],Ø         ;is square empty?
        jne ala                 ;if not, can't move forward
        mov move[di],246        ;store move
        call forward            ;move forward one square
        mov bl,pos[di]          ;get FROM position
        cmp bl,81               ;is pawn on home square?
        jl fb                   ;if not, can't move two squares
        add bl,236              ;calculate position two squares forward
        cmp board[bx],Ø         ;is square empty?
        jne ala                 ;if not, can't move there
        mov move[di],236        ;store move
        call forward            ;move two squares forward
ala:    mov bl,pos[di]          ;get FROM position
fb:     add bl,247              ;calculate position diagonally left
        cmp board[bx],Ø         ;is enemy piece there?
        jle am                  ;if not, can't move there
        mov move[di],247        ;store move
        call forward            ;move diagonally left
am:     mov bl,pos[di]          ;get FROM position
        add bl,245              ;calculate position diagonally right
        cmp board[bx],Ø         ;is enemy piece there?
        jle an                  ;if not, can't move there
        mov move[di],245        ;store move
```

```
         call forward              ;move diagonally right
an:      ret
;
;knight
;
knight:  mov m[di],Ø               ;initialize counter
         mov bl,Ø
ao:      mov al,n_moves[bx]        ;get a knight move
         mov move[di],al           ;store move
         call forward              ;move it
         inc m[di]                 ;next move
         mov bl,m[di]              ;get counter value
         cmp bl,8                  ;tried all moves?
         jl ao                     ;if not, try next move
         ret
;
;bishop
;
bishop:  mov o[di],4               ;set counter limit
         mov m[di],Ø               ;initialize counter
         jmp ap                    ;generate bishop moves
;
;rook
;
rook:    mov o[di],8               ;set counter limit
         mov m[di],4               ;initialize counter
         jmp ap                    ;generate rook moves
;
;queen
;
queen:   mov o[di],8               ;set counter limit
         mov m[di],Ø               ;initialize counter
ap:      mov bl,m[di]              ;get counter value
```

```
dc:     mov al,q_moves[bx]          ;get a move direction
        mov n[di],al                ;store move direction
ar:     mov move[di],al             ;store move
        call forward                ;move it
        mov al,pos[di]              ;get FROM position
        add al,move[di]             ;calculate TO position
        mov bl,al
        cmp board[bx],Ø             ;is TO square empty?
        jne at                      ;if not, next direction
        mov al,move[di]             ;get move
        add al,n[di]                ;move further in same direction
        jmp ar                      ;try that move
at:     inc m[di]                   ;next direction
        mov bl,m[di]                ;get counter value
        cmp bl,o[di]                ;tried all directions?
        jl dc                       ;if not, try next direction
        ret
;
;king
;
king:   mov m[di],Ø                 ;initialize counter
        mov bl,Ø
aq:     mov al,q_moves[bx]          ;get a king move
        mov move[di],al             ;store move
        call forward                ;move it
        inc m[di]                   ;next move
        mov bl,m[di]                ;get counter value
        cmp bl,8                    ;tried all moves?
        jl aq                       ;if not, try next move
        ret
chess   endp
code    ends
        end
```

Appendices

A P P E N D I X A

Beginner's Guide to Typing In Programs

The programs for the IBM PC and PCjr published in this book are written in a language known as BASIC. BASIC (Beginner's All-purpose Symbolic Instruction Code) is the most common programming language for home and personal computers. The programs in this book are BASIC listings generated on a printer. By typing the listing *exactly* as it appears and then saving it on disk, you can add the program to your software library.

If you have never typed in a program listing, please observe the following precautions to avoid problems.

First of all, be sure the program listing is the correct one for your computer. Some of the programs in this book will work as printed on both the PC and PCjr; others will work only on the PCjr or only on the PC.

Second, be sure the program will work on your system configuration. The programs in this book will work only on specific configurations. Special requirements can usually be found at the beginning of the articles.

Third, notice which version of BASIC is required. There are several versions of BASIC for IBM personal computers. These versions are upward-compatible with each other—meaning that programs written in the lower versions will work in the upper versions, but not necessarily vice versa.

Both the PC and PCjr have a built-in version of BASIC called Cassette BASIC. This is the simplest BASIC; programs written in Cassette BASIC will work with all other BASICs. The computer automatically enters Cassette BASIC whenever you switch on the machine without a DOS disk in the disk drive (or if you have no disk drive). As the name implies, Cassette BASIC is designed for using tape storage instead of disk storage.

The next higher version is Disk BASIC. It is included on your DOS disk. To load Disk BASIC on a PC, switch on the computer with the DOS disk inserted in the drive (drive A on dual-drive systems). At the DOS prompt A>, type BASIC and press the Enter key.

The next higher version on the PC is called Advanced BASIC, or BASICA. Load BASICA the same way you would load Disk BASIC, except type BASICA at the DOS prompt. Like Disk BASIC, BASICA is included on your DOS disk.

A slightly higher version of BASIC is available for the PCjr: Cartridge BASIC. This is a plug-in ROM cartridge available from IBM at extra cost. Cartridge BASIC is BASICA but has extra commands to take advantage of the PCjr's special features. Just insert Cartridge BASIC and type BASIC at the DOS prompt; then press Enter.

Computers Are Picky

Unlike the English language, which is full of ambiguities, BASIC usually has only one "right way" of stating something. Each letter, symbol, and number is significant. Type the listings *exactly* as they appear. A common mistake is to substitute the letter *O* for the numeral *0* or a lowercase *l* for the numeral *1*. Also, all punctuation must be entered as it appears in the listing. Substituting a comma for a semicolon or omitting a colon can make a big difference. Even blank spaces can be important.

DATA Statements

Some programs may have one or more sections of DATA statements. These lines are especially critical.

If a single number in a DATA statement is mistyped, your computer could lock up, or crash—the screen may go blank and the keyboard may refuse to acknowledge any more commands. The computer cannot be harmed by this. However, to regain control, you may have to reset the machine by pressing Ctrl-Alt-Del or by turning the power switch off and on again. Both actions will erase the BASIC program.

For this reason, *always save a copy of the program before running it for the first time.* If the computer crashes, you can reload the program and check for the error. It's also a good idea to save the program periodically as you type it in.

Know Your Computer

You should familiarize yourself with your computer before trying to type in a program. Learn the commands for saving and loading programs. Learn to use the keyboard editing functions. You could retype a line if you make a mistake, but it's much easier to insert and delete characters with the Ins and Del keys.

APPENDIX B

The Automatic Proofreader

Charles Brannon

Now there's a way to banish practically all typing errors when entering programs in this book. "The Automatic Proofreader" instantly checks your typing as you enter each line. The Proofreader works on any IBM PC or Enhanced Model PCjr with Cartridge BASIC.

We all know it's hard to type in a program correctly the first time. Seemingly trivial typing errors can trigger dreaded error messages or even a *system crash* (the computer "locks up" and the keyboard won't respond). Usually, the only way to recover from such a crash is to reset the computer by turning it off, then on again—wiping out the memory and all your typing in the process.

Even when you locate and correct the mistyped lines, there always seem to be more, lurking somewhere in the hundred-odd lines of the program. Sometimes you feel like giving up.

Elusive Errors

Some errors are almost impossible to catch, especially if you know little or nothing about BASIC programming. For instance, can you spot the mistake in this line?

```
100 C=C+LEN(STR$(VAL(L$))+1
```

Here's how it should read:

```
100 C=C+LEN(STR$(VAL(L$)))+1
```

Did you catch the difference? A right parenthesis was missing before the +1. (A left parenthesis must always have a matching right parenthesis. If you add up all the parentheses in a statement, you should get an even number.)

An Impossible Dream?

The strong point of computers is that they excel at tedious, exacting tasks. So why not get your computer to check your typing for you? An impossible dream?

Not any more—not with "The Automatic Proofreader." The Automatic Proofreader is a BASIC program that mimics

the IBM BASIC program editor, although it's a little slower. In effect, with the Proofreader you'll be using a BASIC program to enter other BASIC programs.

The Proofreader lets you enter program lines as you normally do, but with an important difference. After you type in a line and press Enter, a pair of letters appears, inserted just before the line you've typed. This pair of letters is called a *checksum*. You compare the checksum to a matching code of letters in the program listing. *If the pair of letters on your screen matches the pair of letters in the program listing, the line was entered correctly.* A glance is all it takes to confirm that you've typed the line right.

Does it sound too good to be true? It isn't. Thousands of readers of our books, *COMPUTE!* magazine, and *COMPUTE!'s Gazette* have been successfully using similar Proofreaders to type in program listings for their Commodore, Atari, and IBM computers.

Using the Proofreader

To get started, type in the Automatic Proofreader listing at the end of this appendix and save a couple of copies. You'll want to use it whenever you enter a program from this book, *COMPUTE!* magazine, and other COMPUTE! books. Naturally, the Proofreader can't check itself, so you'll have to be extra careful when typing it in. Often, when readers experience difficulty with the Proofreader, the problem has been traced to improperly typing the Proofreader program. So check every line very carefully—if you get it right, it'll probably be the last program listing in this book you'll ever have to scrutinize that closely.

When you run the Automatic Proofreader, the screen clears to white and the prompt "Proofreader Ready" appears. At this point, the Proofreader is ready to accept program lines or commands. You can just type in a program as you normally would.

Here's an example of how it works. Type in the following line:

120 RESUME 130

When you press Enter, there'll be a short delay and the checksum will appear:

BE 120 RESUME 130

The two letters *BE* are the checksum. Try making a change in

the line, then press Enter. Notice that the checksum has changed. The slightest alteration to the line results in a different checksum.

All the program listings published in this book have a checksum printed to the left of each line number. Just type in each line (omitting the printed checksum, of course), and compare the checksum on your screen with the checksum in the listing. If they match, go on to the next line. If they don't match, there's a difference between the way you typed the line and the way it appears in the book. It might be a very slight difference that's hard to spot at first. When you find it, you can correct the line immediately with the cursor and editing keys instead of waiting to find the error when you run the program.

Although the Proofreader is an indispensable aid, there are a few things to watch for. First, the Proofreader is *very* literal: It looks at the individual characters in a line. It makes a distinction between uppercase and lowercase, so be sure to *leave Caps Lock on* while you type in a listing, releasing it when necessary to enter lowercase. For similar reasons, do not use ? as an abbreviation for PRINT—they're not the same thing in the Proofreader's eyes. Neither are the numeral *1* and the lowercase *l* and the uppercase *I*, or the numeral *0* and the uppercase *O*. The Proofreader can even catch transposition errors—such as PIRNT instead of PRINT.

The Proofreader is also picky with spaces since proper spacing is important to prevent SYNTAX ERRORs in IBM BASIC. Adding an extra space or leaving one out—even in places where it's normally permitted, such as within PRINT statements or REMarks—will result in a different checksum. If you want to modify something, we recommend that you first type the program exactly as it's published and verify that it runs *before* making your modifications.

Proofreader Commands

The Proofreader has many commands, almost identical in syntax to those found in IBM BASIC. In fact, the editing environment is so similar that you may forget you're using a BASIC program to enter other BASIC programs. If in doubt, remember that BASIC's prompt is "OK" while the Proofreader says "Proofreader Ready." Also, the screen is white when the Proofreader is active.

LIST works just like it does in BASIC. LIST 10 lists just line 10. LIST 40–90 displays all lines between 40 and 90, inclusive. LIST 100– gives you all the lines from line 100 to the end of your program. If you press any key while the program is LISTing, the listing will stop. Unless you are running the Proofreader under PCjr Cartridge BASIC or BASICA 2.0 or 2.1, do not press Ctrl-Break to stop the listing or you will exit the Proofreader. The Break key is trapped with Advanced BASIC 2.0 or 2.1, so you'll get the message "Stopped."

CHECK is a special Proofreader command that acts like LIST, except it also displays the checksum for each line.

LLIST will list the program to the current printer device. It works as LLIST does in BASIC.

NEW clears out the program in memory—not the Proofreader, but the program you're typing. However, there's an extra safeguard built in. Unlike BASIC, the Proofreader will ask, "Erase program—Are you sure?" You must enter Y to erase the program. Remember, this won't remove the Proofreader itself, but only the program held by the Proofreader.

FILES lists the disk directory on the screen. It lists only the directory for drive A.

BASIC exits the Proofreader. It returns you to BASIC's "OK" prompt and returns the screen color to black, leaving the Proofreader still in memory. To be safe, always save your program on disk before leaving the Proofreader. If you accidentally exit the Proofreader by typing BASIC, you can reenter the Proofreader and retrieve your program by typing CONT. You'll get a SYNTAX ERROR message and the screen won't return to white, but the program you were typing will be intact.

SAVE and LOAD Commands

You can save a program at any point when using the Proofreader. In fact, it's a good idea to save a program occasionally as you're typing so an unexpected power failure won't wipe out all your work. Just type SAVE *"filename"*. As usual, the ending quote is optional. If you don't enter a period and a three-character extender, the extender .BAS will be automatically appended. Again, this is just like IBM BASIC.

Unlike IBM BASIC, the Proofreader always saves programs to disk in ASCII form. You can load this program from BASIC like any other. Since ASCII files take up more room on

a disk than ordinary program files, later you may want to resave the program back to disk from BASIC in order to conserve disk space.

You can reload programs into the Proofreader with the command LOAD *"filename"*. (As with the SAVE command, the extender .BAS is assumed if you don't enter an extension.) This way you can type in part of a long program, save it on disk, and load it again later to continue typing. But make sure the program you're loading was saved by the Proofreader. The Proofreader cannot successfully load a program file that's not in ASCII form.

Checksum programs are not new. But unlike other checksum programs, the Automatic Proofreader shows you *instantly,* as soon as you've entered the line, if you've made a typo. We hope that the Proofreader makes your program entry both faster and easier, and that you'll never have to face another frustrating error message again.

The Automatic Proofreader

```
LA 10 'Automatic Proofreader Version 2.00 (Lines
      270,510,515,517,620,630 changed from V1.0)
LD 100 DIM L$(500),LNUM(500):COLOR 0,7,7:KEY OFF:
       CLS:MAX=0:LNUM(0)=65536!
PK 110 ON ERROR GOTO 120:KEY 15,CHR$(4)+CHR$(70):
       ON KEY(15) GOSUB 640:KEY (15) ON:GOTO 130
BE 120 RESUME 130
BJ 130 DEF SEG=&H40:W=PEEK(&H4A)
IH 140 ON ERROR GOTO 650:PRINT:PRINT"Proofreader
       Ready."
KB 150 LINE INPUT L$:Y=CSRLIN-INT(LEN(L$)/W)-1:LO
       CATE Y,1
CA 160 DEF SEG=0:POKE 1050,30:POKE 1052,34:POKE 1
       054,0:POKE 1055,79:POKE 1056,13:POKE 1057,
       28:LINE INPUT L$:DEF SEG:IF L$="" THEN 150
BC 170 IF LEFT$(L$,1)=" " THEN L$=MID$(L$,2):GOTO
        170
NN 180 IF VAL(LEFT$(L$,2))=0 AND MID$(L$,3,1)=" "
        THEN L$=MID$(L$,4)
DA 190 LNUM=VAL(L$):TEXT$=MID$(L$,LEN(STR$(LNUM))
       +1)
ND 200 IF ASC(L$)>57 THEN 260 'no line number, th
       erefore command
OG 210 IF TEXT$="" THEN GOSUB 540:IF LNUM=LNUM(P)
        THEN GOSUB 560:GOTO 150 ELSE 150
```

```
MB 220 CKSUM=0:FOR I=1 TO LEN(L$):CKSUM=(CKSUM+AS
       C(MID$(L$,I))*I) AND 255:NEXT:LOCATE Y,1:P
       RINT CHR$(65+CKSUM/16)+CHR$(65+(CKSUM AND
       15))+" "+L$
JE 230 GOSUB 540:IF LNUM(P)=LNUM THEN L$(P)=TEXT$
       :GOTO 150 'replace line
CL 240 GOSUB 580:GOTO 150 'insert the line
AD 260 TEXT$="":FOR I=1 TO LEN(L$):A=ASC(MID$(L$,
       I)):TEXT$=TEXT$+CHR$(A+32*(A>96 AND A<123)
       ):NEXT
LP 270 DELIMITER=INSTR(TEXT$," "):COMMAND$=TEXT$:
       ARG$="":IF DELIMITER THEN COMMAND$=LEFT$(T
       EXT$,DELIMITER-1):ARG$=MID$(TEXT$,DELIMITE
       R+1) ELSE DELIMITER=INSTR(TEXT$,CHR$(34)):
       IF DELIMITER THEN COMMAND$=LEFT$(TEXT$,DEL
       IMITER-1):ARG$=MID$(TEXT$,DELIMITER)
FC 280 IF COMMAND$<>"LIST" THEN 410
ID 290 OPEN "scrn:" FOR OUTPUT AS #1
LH 300 IF ARG$="" THEN FIRST=0:P=MAX-1:GOTO 340
IJ 310 DELIMITER=INSTR(ARG$,"-"):IF DELIMITER=0 T
       HEN LNUM=VAL(ARG$):GOSUB 540:FIRST=P:GOTO
       340
BP 320 FIRST=VAL(LEFT$(ARG$,DELIMITER)):LAST=VAL(
       MID$(ARG$,DELIMITER+1))
EC 330 LNUM=FIRST:GOSUB 540:FIRST=P:LNUM=LAST:GOS
       UB 540:IF P=0 THEN P=MAX-1
GD 340 FOR X=FIRST TO P:N$=MID$(STR$(LNUM(X)),2)+
       " "
KA 350 IF CKFLAG=0 THEN A$="":GOTO 370
PF 360 CKSUM=0:A$=N$+L$(X):FOR I=1 TO LEN(A$):CKS
       UM=(CKSUM+ASC(MID$(A$,I))*I) AND 255:NEXT:
       A$=CHR$(65+CKSUM/16)+CHR$(65+(CKSUM AND 15
       ))+" "
DO 370 PRINT #1,A$+N$+L$(X)
JJ 380 IF INKEY$<>"" THEN X=P
OF 390 NEXT :CLOSE #1:CKFLAG=0
CA 400 GOTO 130
PD 410 IF COMMAND$="LLIST" THEN OPEN "lpt1:" FOR
       OUTPUT AS #1:GOTO 300
GM 420 IF COMMAND$="CHECK" THEN CKFLAG=1:GOTO 290
KA 430 IF COMMAND$<>"SAVE" THEN 450
CL 440 GOSUB 600:OPEN ARG$ FOR OUTPUT AS #1:ARG$=
       "":GOTO 300
OE 450 IF COMMAND$<>"LOAD" THEN 490
PG 460 GOSUB 600:OPEN ARG$ FOR INPUT AS #1:MAX=0:
       P=0
CH 470 WHILE NOT EOF(1):LINE INPUT #1,L$:LNUM(P)=
       VAL(L$):L$(P)=MID$(L$,LEN(STR$(VAL(L$)))+1
       ):P=P+1:WEND
KK 480 MAX=P:CLOSE #1:GOTO 130
```

```
OG 490 IF COMMAND$="NEW" THEN INPUT "Erase progra
       m - Are you sure";L$:IF LEFT$(L$,1)="y" OR
        LEFT$(L$,1)="Y" THEN MAX=0:GOTO 130:ELSE
       130
CL 500 IF COMMAND$="BASIC" THEN COLOR 7,0,0:ON ER
       ROR GOTO 0:CLS:END
NC 510 IF COMMAND$<>"FILES" THEN 520
IH 515 IF ARG$="" THEN ARG$="A:" ELSE SEL=1:GOSUB
        600
IO 517 FILES ARG$:GOTO 130
DD 520 PRINT"Syntax error":GOTO 130
BO 540 P=0:WHILE LNUM>LNUM(P) AND P<MAX:P=P+1:WEN
       D:RETURN
KM 560 MAX=MAX-1:FOR X=P TO MAX:LNUM(X)=LNUM(X+1)
       :L$(X)=L$(X+1):NEXT:RETURN
GK 580 MAX=MAX+1:FOR X=MAX TO P+1 STEP -1:LNUM(X)
       =LNUM(X-1):L$(X)=L$(X-1):NEXT:L$(P)=TEXT$:
       LNUM(P)=LNUM:RETURN
GA 600 IF LEFT$(ARG$,1)<>CHR$(34) THEN 520 ELSE A
       RG$=MID$(ARG$,2)
EE 610 IF RIGHT$(ARG$,1)=CHR$(34) THEN ARG$=LEFT$
       (ARG$,LEN(ARG$)-1)
LA 620 IF SEL=0 AND INSTR(ARG$,".")=0 THEN ARG$=A
       RG$+".BAS"
DD 630 SEL=0:RETURN
HM 640 CLOSE #1:CKFLAG=0:PRINT"Stopped.":RETURN 1
       50
II 650 PRINT "Error #";ERR:RESUME 150
```

APPENDIX C

Disk Instructions

Typing in long BASIC programs can be a time-consuming task. As a service to our readers, COMPUTE! Publications has made available for purchase a disk which contains all the programs in this book. If you wish to purchase *COMPUTE!'s First Book of IBM* disk, use the coupon found in the back of this book or call toll-free 1-800-334-0868.

Preparing a Purchased Disk

If you have purchased *COMPUTE!'s First Book of IBM* disk and simply try to insert it in your drive and turn on the computer, you'll get the following message:

Non-System disk or disk error

The disk you purchased doesn't contain the IBM system files necessary to start the IBM. You have two options. You can boot the system each time with a disk that does contain COMMAND.COM and BASICA (BASICA is not necessary with the PCjr), or you can copy the system files to your program disk by following the directions below for your system.

Single-Drive Systems

1. As a precaution, cover the notch on the PC-DOS master disk (or a copy of the master disk) with one of the write-protection tabs that come with each box of disks. This will prevent you from accidently writing data to the wrong disk.
2. Turn on your IBM with a PC-DOS master disk inserted in the disk drive.
3. Enter this command at the A> prompt:

   SYS B: (and press Enter)
4. You will have to swap disks a few times. You should insert the PC-DOS master disk in the disk drive whenever you see the message:

   Insert diskette in drive A: and strike
   any key when ready

   and insert the *COMPUTE!'s First Book of IBM* disk whenever you see the message:

Insert diskette in drive B: and strike any key when ready

Be sure you insert the correct disk each time.

5. Enter the following command at the A> prompt, swapping disks as necessary:

   COPY COMMAND.COM B: (and press Enter)

6. If you're using a PC you'll also need to copy BASICA or BASIC. After entering the following command at the A> prompt, insert a disk that has BASICA or BASIC on it when prompted for drive A, and insert the *COMPUTE!'s First Book of IBM* disk when prompted for drive B.

   COPY BASICA.COM B:

   or

   COPY BASIC.COM B:

   You're all ready to use the disk.

Double-Drive Systems

1. As a precaution cover the notch on the PC-DOS master disk (or a copy of the master disk) with one of the write-protection tabs that come with each box of disks. This will prevent you from accidently writing data to the wrong disk.
2. Turn on your IBM with a PC-DOS master disk inserted in the disk drive A and the *COMPUTE!'s First Book of IBM* disk in drive B.
3. Enter this command at the A> prompt:

   SYS B: (and press Enter)

4. Enter this command at the A>:

   COPY COMMAND.COM B: (and press Enter)

5. If you're using a PC, you'll also need to copy BASICA or BASIC. Insert a disk that has BASICA or BASIC on it in drive A. Enter this at the A> prompt:

   COPY BASICA.COM B:

   or

   COPY BASIC.COM B:

You're all ready to use the disk.

Using the *COMPUTE!'s First Book of IBM* Disk

Once you have added the system files, your disk will load and run automatically. Insert the *COMPUTE!'s First Book of IBM* disk in drive A. If your computer is still on, hold down the Alt and Ctrl keys while pressing the Del key; if your computer is off, turn it on. Once the system is booted, follow the screen instructions to select the program you wish to run.

Preparing a Blank Disk

Warning: The instructions listed below are **not** for use with *COMPUTE!'s First Book of IBM* disk that can be purchased from COMPUTE! Publications. If you have purchased a disk, see the instructions that begin at the heading "Preparing a Purchased Disk" above.

If you have chosen to type in the programs, the first step is to format a disk. You'll need a blank double-sided, double-density, 5-1/4-inch floppy disk. (Although single-sided disks are also acceptable, we recommend double-sided. The Format command listed below is for double-sided disks.) Be sure that you use a blank disk since *formatting a disk erases any data that might be on the disk.* Insert your PC-DOS master disk (or a copy of the master disk) into drive A. Enter the date and time when asked.

At the A> prompt enter the following command:

FORMAT B:/S/V (and press Enter)

The drive will spin and then you'll see:

Insert new diskette for drive B:
and strike any key when ready

Now, insert the blank disk in drive B if you have a dual-drive system, or replace the DOS disk in drive A with a blank disk if you have a single-drive system; press any key. When the formatting is complete you'll be prompted for a volume name; enter a name for this disk.

Since most of the programs in this book are written in BASIC, we suggest that you copy BASICA.COM (or BASIC.COM) to this disk (copying BASICA is not necessary for the PCjr since the Enhanced PCjr has BASIC on a cartridge). Insert a disk containing BASICA.COM into drive A

and your newly formatted disk in drive B (if you use a single-drive system you'll be prompted to insert the new disk). Enter the following at the A> prompt:

COPY BASICA.COM B: (and press Enter)

or

COPY BASIC.COM B:

To begin typing in a BASIC program, enter:

BASICA (and press Enter)

You are now ready to type in and save any of the programs in *COMPUTE!'s First Book of IBM*. We suggest you read Appendices A and B and type in and save "The Automatic Proofreader" first.

Index

Whether you use your IBM PC or PCjr at home or at the office, whether you've owned a PC for years or just bought a new XT, you'll find *COMPUTE!'s First Book of IBM* full of useful programs and articles for every member of the family.

There are games which teach and entertain; tutorials which illustrate programming techniques; applications to help you with your finances; and graphics utilities that can be used to transform your screen into an array of colorful pictures and designs.

COMPUTE!'s First Book of IBM has something for every IBM user, including:

- "Chess," an excellent version where you can play against a friend or the computer (with commented source code for the machine language section).
- "The Screen Machine," which lets you create colorful displays without knowing anything about programming.
- A menu program that loads and runs any BASIC program on a disk with the touch of a few keys. It even allows you to add meaningful descriptions about each file.
- An electronic gradebook that will store and average grades for any teacher.
- "Disk Rx," which lets you easily peek at any sector on a disk.
- Hints on choosing the right type of display screen.
- A program that lets you unlock BASIC programs.
- Educational games like "States and Capital Tutor," "Word Hunt," and "Spelling Bee."

All the programs are complete and ready to type in using "The Automatic Proofreader," a program entry system which virtually makes mistakes obsolete. And, since this is a COMPUTE! publication, you can be certain the writing is clear and the programs easy to use.

ISBN 0-87455-010-6